MSX-Más allá del Basic

Jesús y José L. ASÍN GASCÓN



Bitvision Software - La Coruña 2001

MSX-MÁS ALLÁ DEL BASIC

por Jesús y José L. Asín Gascón

Ctra. de Canillas, 144
28043 Madrid - España
Telefs. 200 97 46/47

I.S.B.N.: 84 - 86381 - 22 - 3
Depósito Legal: M. 35952 - 1986
Imprime: Signo Impresores, S. A.
Albasanz, 27. 28037 Madrid.

Introducción

El libro que tiene en sus manos es fruto de la experiencia acumulada durante casi dos años, cuando las primeras máquinas del Standard Msx hicieron su aparición en el mercado de este país.

Desde el principio nos extrañó la escasa y muchas veces nula información existente sobre el sistema. Afortunadamente la perspectiva ha ido cambiando; pero salvo honrosas excepciones (pocas por cierto) aún se echa de menos un libro específico que incicie en el estudio de la memoria gráfica. Un libro que incite al usuario paso a paso y con ejemplos y trucos harto explicativos a investigar por sí mismo desde una perspectiva clara que, o bien no interesa explicar o nadie se atreve a exponer.

Nosotros, queriendo llenar este vacío, pretendemos recopilar, aquí, parte de lo aprehendido en nuestra andadura por entre los «intrincados laberintos» y chips de un ordenador MSX y descorrer el velo «mágico» que oculta sus secretos.

Pretendemos que el usuario se atreva a «profanar», sin ningún temor, ese lugar y que con imaginación y paciencia aprenda a dirigirlo según sus necesidades.

El libro está pues, dirigido a todos, pero en especial a los «curiosos» que a pesar de la capacidad y potencia del Basic-MSX, este se les quede pequeño y no se atrevan a adentrarse en la «Zona Tenebrosa» del CM por respeto a esa secuencia extraña de números aparentemente imposible de entender.

Si al final, usted es capaz de utilizar pequeñas rutinas en código máquina en estrecha conjunción con el Basic, habrá logrado que éste nunca se le quede pequeño y nosostros habremos cumplido con nuestro objetivo ya que sin necesidad de que usted se adentre de lleno en la maraña del CM, habrá sido capaz de romper la barrera y llegar Más Allá del Basic.

«MAS ALLA DEL BASIC» espera ser un puente hacia el mundo pleno y complejo que es la programación directa en el lenguaje de la CPU.

Con ese sano propósito, queríamos, a modo de ejercicio, que el lector uniera los ejemplos fragmentados a lo largo de la obra para ensamblar un programa comercial, del cual fuimos autores. Lamentablemente, el programa (Detective O. Welles) fue dotado con el segundo premio del concurso MSX de Sony; negándose esa multinacional a otorgar el permiso oportuno para que en todo o en parte fuera publicado en este libro. Sin embargo, gracias a la «generosa negativa» de tan «prestigiosa marca» que ha hecho retrasar la publicación casi seis meses, hemos tenido tiempo de mejorar sensiblemente el esquema del libro ampliando contenidos, ofreciendo ejemplos menos específicos y más útiles, y ganando, en definitiva, profundidad y rigor.

Aunque hemos puesto especial cuidado en prevenir cualquier tipo de error, especialmente en los listados y rutinas CM, es posible que se haya colado algún fantasma. Pedimos disculpas anticipadas de los errores que directamente se nos puedan imputar y rogamos al lector especial atención a la hora de transcribir los listados para lo cual le aconsejamos que, sobre todo en los de CM, repase los datos concienzudamente antes de la ejecución.

SALAMANCA

Agosto 1986

NOTA

Para facilitar en todo momento, lo más posible, la introducción de los listados del libro, es aconsejable que el lector los teclee en modo de texto y a 40 columnas, ya que así se han sacado por impresora.

Será, también buena costumbre ejecutar el siguiente mandato nada más encender el ordenador:

POKE &HFBB0,6

con lo cual aseguraremos la recuperación de los datos de una rutina CM por «cuelgue» del ordenador, mediante la pulsación simultánea de:

⟨CTRL+SHIFT+CODE+GRAPH⟩

La explicación la encontrará en páginas interiores: Cap. IV.

Contenido

Capítulo 1

Nociones generales

1. Software y Hardware

En el proceso que realiza un ordenador, (tratamiento automático de la información), podemos distinguir dos partes bien diferenciadas que, en palabras sencillas, podrían resumirse en lo que se puede tocar y lo que no, es decir, la parte física del aparato y los programas que utiliza para realizar el proceso de datos.

Hardware se empleará, pués, para designar la parte física del aparato o, literalmente, la circuitería del mismo.

Software se empleará para designar el conjunto de datos contenidos en soportes externos, de forma organizada para facilitar su localización.

SOFTWARE

1. BASICO

A— Programas de control
- Gestión de datos
- Gestión de sistemas
- Gestión de trabajos

B— Programas de proceso
- Traductores
  - Ensambladores
  - Compiladores
  - Interpretes
- De servicios
  - Manipulación de datos
  - Servicios para el sistema

2. DE APLICACIONES

La misión del Software básico sería la de facilitar la comunicación del hombre y la máquina.

2. Lenguajes de programación

El ordenador sólo «comprende» las instrucciones que son introducidas según un determinado formato que, al igual que cualquier otro lenguaje, posee su sintaxis, sus reglas de formación. A esto se le llama lenguaje de programación.

Lenguaje de programación es un conjunto limitado de palabras y símbolos, teniendo normalmente como referencia el código ASCII, y que hacen referencia a datos, operaciones, procedimientos, decisiones, etc... susceptibles de ser ejecutados por el ordenador. Estos lenguajes, habitualmente en inglés, se denominan «Lenguajes de alto nivel» y utilizan una serie limitada de instrucciones que se organizan según unas normas sintácticas muy estrictas. Ejemplos de estos lenguajes serían: BASIC, FORTRAN, PASCAL, ENSAMBLADOR, etc.

Sin embargo, el procesador trabaja en lenguaje máquina, es decir, manejando números digitales, con sólo cifras 0 y 1; por lo tanto, para que el ordenador pueda comprender las instrucciones de un lenguaje de alto nivel, necesita un traductor de lenguaje que genere una secuencia de instrucciones en binario que específicamente caracterice a esas instrucciones, pero no a otras. El traductor puede encontrarse como residente o puede ser cargado desde un dispositivo externo. Si es residente, una mirada a la extensión de la ROM, nos dará una idea de la potencia del lenguaje utilizado.

Teniendo en cuenta el método de traducción, podemos distinguir dos tipos de lenguajes: el Compilador, que traduce globalmente el progama y el Intérprete, que traduce paso a paso en el momento de la ejecución.

El compilador

El compilador es un programa utilizado por lenguajes de alto nivel que permite traducir un programa escrito en estos lenguajes a lenguaje máquina. La operación se llama compilación.

El compilador, también llamado ensamblador, traduce el programa completo antes de que se ejecute. El proceso que realizan sería el siguiente: traducir las instrucciones a CM. y colocarlas unas detrás de otras. Calcular las direcciones relativas de cada etiqueta o campo y dar como resultado un código objetivo. El resultado final no es otro que un nuevo programa, esta vez en CM. pero equivalente al anterior.

El intérprete

El intérprete, a diferencia del compilador, lee cada instrucción de un programa escrito en un lenguaje de alto nivel y la traduce en el momento de la ejecución, realizando el mismo proceso cada vez que el programa se ejecuta. Esta es una de las razones por las que un programa escrito en BASIC tiene un tiempo de ejecución más largo que un programa realizado en CM.

El ensamblador

El lenguaje máquina es el lenguaje de la CPU. y por tanto el de más bajo nivel, es decir, el más difícil para nosotros ya que está orientado hacia la máquina, y ésta sólo funciona con cifras. Para solventar esta dificultad, los primeros programadores idearon un lenguaje intermedio capaz de hacer más claros e inteligibles los programas escritos en CM. Este lenguaje se denominó Ensamblador, en inglés, Assembler. Después, como hemos visto, un compilador se encargará de generar automáticamente su equivalente en CM. Ese programa compilador se llama también Ensamblador.

Volveremos más adelante sobre ello puesto que este libro pretende, entre otros objetivos, enseñarle a programar en Ensamblador.

3. Unidades de información

La unidad elemental de información es llamada bit; abreviatura de la palabra inglesa "binary digit", o dígito binario.

Se puede pensar que un bit es una lámpara que puede estar encendida o apagada en un determinado tiempo; de esta manera, podríamos representar cuando una lámpara está encendida mendiante el símbolo 1 y cuando está apagada mediante el 0.

Así pues, un bit es igual a una decisión binaria o la designación de uno de los dos posibles valores o estados. La información se representa típicamente mediante determinadas series de bits, ya que con un bit sólo podríamos representar un estado (1 o 0).

Los ordenadores MSX tienen un procesador de 8 bits, esto quiere decir que cada casilla de memoria podrá almacenar un número comprendido entre los valores decimales 0 y 255.

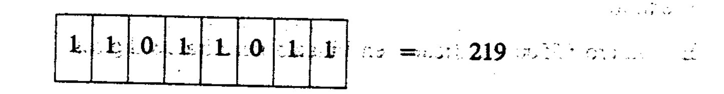
En efecto: descompongamos un bloque de ocho bits, es decir, un byte y numeremos de derecha a izquierda los bits, partiendo desde el cero.

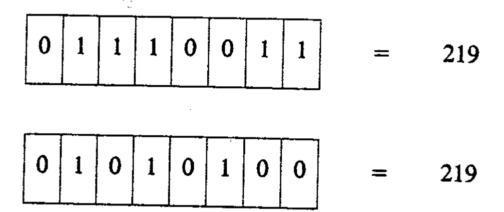
Ahora pidamos al ordenador, en modo directo, que calcule la siguiente operación:

```
?1*2^7+1*2^6+1*2^5+1*2^4+1*2^3+1*2^2+1*2^1+1*2^0
```

Su ordenador organiza la información en bloques de ocho bits contiguos y que ocupan una sola posición de memoria. Un byte podrá contener, por tanto, cualquiera de las 256 combinaciones posibles de ocho dígitos, es decir, 2^8.

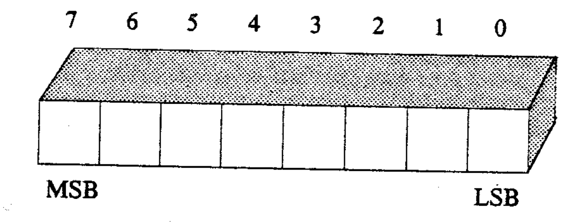
Ejemplos:





Una palabra es el número de bits que puede manejar un ordenador simultáneamente. Si el número de bits en una palabra es igual a ocho, utilizaremos indistintamente byte o palabra.

El microprocesador Z-80 tiene una longitud de palabra de ocho bits.



De los ocho bits que componen una palabra, el bit 7 es considerado como el más significativo MSB; y el menos significativo, LSB, el bit 0.

4. Dirección de memoria

Definimos dirección de memoria como la posición de almacenamiento de una palabra en memoria.

El chip del microprocesador Z-80 puede direccionar 65536 (64K) posiciones de memoria distintas, conteniendo cada una de ellas ocho bits. El chip contiene 16 bits de palabra de direccionamiento. Por tanto, si se realiza un cálculo simple tendremos que 2^16 equivale a todas las posibles posiciones de memoria (65536).

Si antes decíamos que con un byte podíamos expresar un número decimal del 0 al 255, ¿cómo podemos pedir al ordenador que mire, por ejemplo, el contenido de una dirección de memoria mas allá de la 50000, o que cargue números superiores a 255?

El Z-80 trata las direcciones de memoria de 16 bits como dos bytes de dirección de memoria, un byte alto HI-byte de ocho bits y un byte bajo LO-byte de ocho bits.

Ejemplo:

El número 65500 codificado en binario necesita 16 dígitos

1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0

Los ocho más significativos (los más a la izquierda.) corresponderán al HI-byte y los ocho menos significativos al LO-byte. De esta manera es posible cubrir toda la posible gama de posiciones de memoria de lectura/escritura.

Recordaremos siempre que para especificar una posición de memoria deberemos calcular los dos bytes de dirección que juntos y contiguos expresarán una dirección de memoria de 16 bits.

Para calcular la parte alta dividiremos la dirección entre 256 y nos quedaremos con la parte entera. El resto de la división será el LO-byte o la parte baja.

Ejemplos:

Calcular la parte alta y baja de 44458, (&HADAA).

Dividimos 44458 entre 256 y nos quedamos con la parte entera, 173. Después a 44458 le restamos la parte entera multiplicada por 256. El resto, 170, será la parte baja. En hexadecimal: parte alta &HAD y parte baja &HAA.

Sin embargo, tenga siempre presente que para codificar en CM escribiremos siempre el byte bajo en una posición de memoria inmediatamente anterior a la casilla que ocupe el byte alto. De tal manera que si consideramos el ejemplo anterior como una dirección de memoria a la que tiene que saltar el programa y queremos escribirla a partir de la dirección 34000, (&H84D8), escribiremos en &H84D8 el byte bajo, &HAA, y en &H84D8+1, la parte alta, es decir, &HAD.

Recuerde ésto último porque es muy importante.

Al final del presene capítulo encontrará usted ejercicios para que practique estos conceptos. Si no se quiere molestar también le diremos el método rápido para que su propio ordenador se las proporcione directamente en hexadecimal. Por esta y otras razones que irá viendo compartirá con nosotros la comodidad del sistema hexadecimal en programación.

5. Sistemas de numeración

A lo largo de la historia de la humanidad, el hombre ha intentado la creación de diversos sistemas para contar y numerar objetos. El más extendido de ellos es el decimal. En este sistema, debido sin duda a que los primeros hombres empezaron a contar con los dedos de las manos, disponemos de una combinación de diez dígitos distintos, desde el 0 hasta el 9, dispuestos de derecha a izquierda en unidades, decenas, etc.

Si quisieramos descomponer un número decimal, por ejemplo el 7867, tendríamos:

7 Unidades	=	7*1	=	7
6 Decenas	=	6*10	=	60
8 Centenas	=	8*100	=	800
7 U.Mil	=	7*1000	=	7000
				7867

Cada signo tiene un valor y el valor del número se halla multiplicando el valor de cada uno de ellos por 10 elevado a la potencia correspondiente a la posición que ocupe en el número.

Así, en el caso que nos ocupa, tendríamos:

$$7867 = 7*10^0+6*10^1+8*10^2+7*10^3$$

Este sistema se denomina de base 10 porque la base de sus exponentes es siempre 10. Pero ¿qué habría sucedido si los hombres en vez de tener diez dedos hubiéramos tenido sólamente dos? Pues que muy posiblemente calcularíamos en una base distinta.

El ordenador sólo dispone, por así decir, de dos dedos, el cero y el uno, porque sus componentes sólo admiten dos estados: encendido y apagado; paso de corriente, no paso de corriente. Funciona, pues, con un código digital binario.

Veamos cómo lo hace.

El código binario

Es el código digital más simple. Está formado, como ya hemos visto anteriormente por dos estados: el 0 y el 1.

Asociados a estos dos estados podremos representar los números equivalentes a los cuatro primeros decimales:

BIN	DEC
0 0	0
0 1	1
1 0	2
1 1	3

Si aumentamos el número de dígitos al doble, tendríamos la posibilidad de codificar 16 números decimales con el cero incluido, es decir 2^4.

Veamos:

	BIN	DEC
Pesos	8 4 2 1	
	0 0 0 0	0
	0 0 0 1	1
	0 0 1 0	2
	0 0 1 1	3
	0 1 0 0	4
	0 1 0 1	5
	0 1 1 0	6
	0 1 1 1	7

	BIN	DEC
Pesos	8 4 2 1	
	1 0 0 0	8
	1 0 0 1	9
	1 0 1 0	10
	1 0 1 1	11
	1 1 0 0	12
	1 1 0 1	13
	1 1 1 0	14
	1 1 1 1	15

Siguiendo este mismo procedimiento, un número binario de 8 bits podrá codificar 256 números decimales diferentes, que no importa lo que puedan ser (datos, códigos, instrucciones, etc.)

El valor de un número binario se calcula de la misma manera que hemos visto para el decimal, pero cambiando la base 10 por la base en la que operamos.

La codificación hexadecimal

¿Podría usted recordar la siguiente lista de números de ocho bits en un segundo?

01001110
10110101
11010101
00100101

Seguramente deducirá usted la posibilidad de que exista un método más asequible para recordar números binarios de 8 bits.

En efecto, existe un sistema de contaje hexadecimal, que como dice la palabra, cuenta en base 16 y que consta de 16 signos diferentes, los diez decimales y otros 6, de la A a la F, para completar los 16: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

De esta forma tendríamos para los 16 primeros números la siguiente tabla:

BIN	HEX	DEC
0 0 0 0	00	0
0 0 0 1	01	1
0 0 1 0	02	2
0 0 1 1	03	3
0 1 0 0	04	4
0 1 0 1	05	5
0 1 1 0	06	6
0 1 1 1	07	7
1 0 0 0	08	8
1 0 0 1	09	9
1 0 1 0	0A	10
1 0 1 1	0B	11
1 1 0 0	0C	12
1 1 0 1	0D	13
1 1 1 0	0E	14
1 1 1 1	0F	15

La representación hexadecimal se utiliza mucho en ensamblador, como veremos, ya que la unidad de base que sirve para todos los cálculos es el número de ocho bits. Este sistema permite representar cualquier registro, o cualquier celdilla de memoria por dos dígitos hexadecimales.

Si ampliamos la tabla que antes ofreciamos, comprenderemos perfectamente la ventaja del sistema hexadecimal.

BIN.	HEX.	DEC.
0001 0000	10	16
0001 0001	11	17
0001 0010	12	18
0001 0011	13	19
0001 0100	14	20
0001 0101	15	21
0001 0110	16	22
0001 0111	17	23
0001 1000	18	24
0001 1001	19	25
0001 1010	1A	26
0001 1011	1B	27
0001 1100	1C	28
0001 1101	1D	29
0001 1110	1E	30
0001 1111	1F	31

Hemos agrupado los números de 8 bits en grupos de 4, para comprender mejor como se realiza el proceso de conversión de un número hexadecimal a binario, y para leer mejor el número binario.

Para convertir un número binario de 8 bits en el código hexadecimal, necesitamos tres pasos:

- Escribir el número binario de ocho bits completo, rellenando de ceros por la izquierda, si hace falta
- Hacer dos grupos de cuatro dígitos binarios.
- Sustituirlos por el dígito hexadecimal correspondiente para cada grupo.

Ejemplo:

Codificar en hexadecimal el n.° 10111111

- Separamos: 1011 1111
- Sustituimos: B F
- Resultado: BF&H

En lo sucesivo nos acostumbraremos a anotar &H para los números en hexadecimal y &B para los binarios. Los que no vayan seguidos de notación especial se considerarán decimales.

Cambio de bases

Para pasar un número de base decimal a cualquier otra base, se divide el número por la base, sin sacar decimales. El resto será el primer dígito derecho del número buscado. Después se realiza sucesivamente la misma operación hasta obtener un cociente cero. Entonces escribiremos todos los restos, empezando por el último y los escribiremos de izquierda a derecha.

Ejemplo:

Calcule el equivalente binario del n.° 16 decimal.

```
16 |_2___
 0  8 |_2___
    0  4 |_2___
       0  2 |_2___
          0  1 |_2___
             1  0
```

El número resultante será igual a los restos sucesivos. Es decir: 1 0 0 0 0

Ahora pase a hexadecimal el n.° 110 decimal.

```
110 |_16___
 14   6 |_16___
      6   0
```

El número buscado será 6E&H ya que 14 = E&H.

Es muy importante saber trabajar con las distintas representaciones numéricas, por eso antes de continuar le proponemos los siguientes ejercicios:

Codificar en hexadecimal los siguientes números decimales:

63456	23441	12876
10021	20034	465

Codificar en hexadecimal los siguientes números binarios:

&B 1110 0010 &B 0101 1101 &B 1011 0110

Codificar en decimal los siguientes números binarios:

&B 1110 0110 &B 1000 0100 &B 0001 0011

Codificar en binario los siguientes números hexadecimales:

&H4DCC &HFA16 &HACDC &HBA1F &HE9AF

Operaciones con números binarios

La Suma

Lo primero que debemos tener presente es la tabla de sumar que por lo demás, es sencillísima:

0+0=0	1+0=1
0+1=1	1+1=10

La suma se efectúa como en base decimal, pero empleando la tabla binaria para la suma. Cuando sumemos 1+1, escribiremos 0 y nos llevaremos una unidad de orden superior.

Ejemplo:

```
  0 1 0 1 0 1 0 1
  0 0 1 1 0 1 1 0
  ---------------
  1 0 0 0 1 0 1 1
```

Representación binaria en complemento a dos

La representación en complemento a dos es una forma de codificar números enteros similar a la binaria. La ventaja fundamental es que podemos representar con ella los números enteros negativos.

Veamos:

BIN. de 4 bits en comp. a dos	n.º decimal
0 1 1 1	7
0 1 1 0	6
0 1 0 1	5
0 1 0 0	4
0 1 1 1	3
0 0 1 0	2
0 0 0 1	1
1 1 1 1	−1
1 1 1 0	−2
1 1 0 1	−3
1 1 0 0	−4
1 0 1 1	−5
1 0 1 0	−6
1 0 0 1	−7
1 0 0 0	−8

De esta especial configuración podemos hacer las deduciones siguientes:

Con cuatro bits podíamos escribir desde el 0 al 15, ambos incluidos. Con complemento a dos podemos escribir desde −8 hasta +7, siendo la mitad los códigos positivos y la otra mitad los negativos.

Los números positivos en complemento a 2 tienen el bit de signo a cero (el más significativo) mientras que los negativos comienzan por 1.

Dos números en complemento a dos se complementan uno al otro, es decir, sumados dan cero.

Veamos:

```
  0 0 0 1      0 0 1 0      0 0 1 1
  1 1 1 1      1 1 1 0      1 1 0 1
 ---------    ---------    ---------
 1 0 0 0 0    1 0 0 0 0    1 0 0 0 0
```

En efecto. Sin embargo, notaremos que el resultado de cada suma tiene un bit más que los operandos. Este bit se denomina acarreo y se pierde; por lo tanto el resultado de cada suma que hemos efectuado es cero.

Para calcular el complemento a dos de un número deberemos realizar las siguientes operaciones:

- Cambiar todos los números que están 0 lógico a 1 lógico y al revés.
- Al número resultante le sumamos 1
- Si sale acarreo, éste se pierde.

Ejemplo: calcular el complemento a dos de 0 1 0 0 &B

- Invertimos cada bit del número — 1 0 1 1
- Le sumamos 1 — 0 0 0 1

```
  1 0 1 1
  0 0 0 1
  -------
  1 1 0 0
```

Si sumamos, ahora el número inicial y su complemento a dos deberá dar como resultado cero.

Veamos:

```
   0 1 0 0
   1 1 0 0
  ---------
  1 0 0 0 0
```

En efecto, el resultado es cero puesto que el acarreo se pierde.

6. El ensamblador del Z-80

Cuando se trata de programar en código máquina o simplemente de realizar rutinas de utilidad, resulta muy útil disponer de un intermediario entre el lenguaje del ordenador y el programador. El lenguaje ensamblador representa a este intermediario,

ya que sería muy duro traducir a "mano" los programas en código máquina.

El ensamblador es un programa de "ayuda" que realiza la traducción de un texto escrito en ensamblador a los correspondientes códigos de operación que utilice el microprocesador de que se trate. En nuestro caso para el Z-80.

Este lenguaje, a diferencia del intérprete "compila" de una sola vez el programa realizado en ensamblador, generando otro programa equivalente, pero esta vez en código máquina.

El programa escrito en ensamblador se denomina programa fuente y al generado código objeto.

Al igual que cualquier otro lenguaje, éste se rige por unas normas sencillas pero estrictas que deberán ser respetadas antes de realizar la "compilación", ya que de lo contrario se generará un código de error.

Sintaxis del ensamblador

Un programa en ensamblador se compone, como es lógico, de una serie de líneas numeradas consecutivamente y en cada una de ellas una instrucción que sea aceptada por el ensamblador de que se trate.

A diferencia del BASIC, cada línea tiene una estructura previamente definida y el intentar alterarla tiene su correspondiente código de error.

Cada línea está dividida en campos. Cada campo tiene una longitud máxima predefinida; y cada uno de ellos está separado por un delimitador que normalmente es un espacio.

Campo 1: Etiqueta

Una etiqueta "label" es un símbolo que se emplea por el programador para representar ciertos datos, o direcciones, saltos, etc. dentro del programa. El ensamblador los mantiene en línea de programa para ayuda del programador, pero en realidad, el "ensamblado" posterior del programa calculará automáticamente la dirección de la instrucción "marcada" por la etiqueta.

Ejemplo:

```
10 SUMA: LD A,13
20       LD (61500),A
30 ------
90 RET
-------
200 CALL SUMA
```


La subrutina que empieza en la línea que lleva la etiqueta SUMA será ejecutada cada vez que sea llamada por este nombre a lo largo del programa.

La etiqueta no es un campo obligatorio pero es muy útil tanto en la depuración de un programa como en su realización.

En la formación de una etiqueta podremos usar cualquiera de los caracteres comprendidos entre el 0 y el 9, a y Z, etc, pero el primer carácter debe ser obligatoriamente una letra.

La etiqueta sólo puede ser definida una vez con el mismo nombre.

La definición de etiqueta no admite palabras reservadas. Tampoco es conveniente definirlas con las letras A, B, C, D, E, HL ... puesto que son palabras reservadas a los registros del Z-80.

La longitud de una etiqueta estará limitada a un determinado número de caracteres, dependiendo del ensamblador utilizado. Lo normal son seis u ocho caracteres.

El ensamblador mantiene un contador de posición que permite asociar a las etiquetas las direcciones de memoria correspondientes. Para dar un valor determinado al contador de posición se emplea la seudoperación ORG.

Cuando el ensamblador encuentra una etiqueta la coloca en un archivo de símbolos "Symbol Table" y con cada una de ellas dos punteros encargados de relacionarla con las etiquetas anterior y posterior en orden alfabético.

Campo 2: Código de Operación

En caso de que exista etiqueta deberá ir detrás, separado por un espacio en blanco.

Este campo se reserva para el codope, "Cod. Ope." de una instrucción. El codope deberá ser válido para el "micro" con el que se trabaje; y normalmente vendrá dado por una sucesión de 2, 3 o 4 letras que definan una instrucción ejecutable.

Al final del libro le proporcionamos una tabla con los comandos que "entiende" el Z-80 y sus correspondientes "nemónicos" para ensamblador.

Campo 3: Operandos

Las expresiones utilizadas van a continuación del campo anterior separadas por un "blanco".

Podemos utilizar un término o varios términos separados por operadores.

Un término puede ser: un número decimal, hexadecimal, binario, una constante literal, una etiqueta, etc.

Un operador puede ser: expresiones aritméticas (+,−), lógicas (And, Xor, Or).

Un operando escrito entre paréntesis representa una dirección:

```
LD HL,(50000)
```

significa "cargar HL con el contenido de la dirección 50000".

Se pueden insertar blancos entre términos y operadores, pero no en medio de un término.

La forma de escribir el operando dependerá del modo de direccionamiento empleado.

Campo 4: Comentarios

Este campo al igual que el de etiqueta es opcional y está separado del resto por un espacio y un ";". Todos los caracteres escritos a continuación serán tomados como simple comentario. Lo cual nos permite documentar el programa. Su función es similar a la sentencia REM del BASIC.

Es posible utilizar líneas enteras como comentario simplemente introduciendo como primer símbolo de línea ";".

Seudoperaciones

Además de los códigos nemónicos correspondientes a las operaciones del Z-80, existe otra serie de códigos que no son propiamente operaciones y que por tanto no generan códigos de operación del Z-80. Solamente son utilizadas por el ensamblador para su uso interno.

Veamos a continuación las más usuales en un ensamblador.

ORG

Se suele colocar al principio del programa y asigna al contador de posición el valor de la expresión. Esto nos permitirá elegir la dirección de memoria a partir de la cual deseamos situar el programa objeto.

EQU expresión

Asigna un valor numérico a la etiqueta y debe comenzar siempre por un nombre de etiqueta.

DEFB expresión, expresión ...

Sirve para almacenar valores en posiciones de memoria. Coloca en la posición de memoria indicada por el contador de posición el valor de la primera expresión; después se incrementa el contador introduciendo el valor de la siguiente expresión y así sucesivamente. Puede ser comprendido con su similitud al DATA del BASIC.

DEFW expresión, expresión ...

Es similar al anterior, pero ahora la longitud de los valores es de 16 bits. Por tanto el contador se incrementará aquí de dos en dos; y su orden de almacenamiento será parte baja, parte alta.

DEFS expresión

Incrementa el contador de posición en una cantidad equivalente al valor de la expresión; es decir, rellena de ceros desde la anterior posición a la indicada por medio de esta seudoperación.

DEFM /literal/

Llena n posiciones de memoria a partir del contador de posición con el código ASCII de la cadena de caracteres incluidas entre comillas.

END

Indica al ensamblador que el texto fuente ha terminado y que las líneas que siguen sean ensambladas normalmente.

7. Instrucciones más usuales del Z-80

No pretendemos en este apartado hacer un estudio exhaustivo de este tema ya que por sí solo merece un libro aparte. Pero sí consideramos necesario hacer un resumen de las instrucciones más utilizadas para que el usuario pueda iniciarse sobradamente en el tema y abordar la resolución de pequeñas rutinas en CM.

Instrucciones de carga

Como su nombre indica sirven para almacenar temporalmente un dato. Salvo algunas muy especiales que no vamos a ver, estas instrucciones no afectan a ninguno de los 8 bits del registro F.

El nemónico es la abreviatura de la palabra inglesa: LoaD -LD-, "cargar".

Dependiendo de la longitud del dato (1 o 2 bytes), podemos distinguir entre instrucciones de carga de 8 bits y de 16 bits.

El formato general se ajusta al siguiente patrón:

LD n,m o LD nn,mm

Donde n siempre será uno de los registros simples; y m o bién otro registro o un dato de ocho bits.

Para las instrucciones de carga de 16 bits, nn puede ser un registro doble o bien una dirección de memoria y mm puede ser un dato o una dirección de memoria. Aquí no está permitida la transferencia entre registros dobles excepto en los siguientes casos:

LD SP,HL
LD SP,IX
LD SP,IY

Los distintos valores de n y m o nn y mm, dan lugar a distintos modos de carga con los que a su vez distinguiremos diversos modos de direccionamiento.

a) LD reg,cte

Este es un sistema de carga directo donde el registro puede ser cualquiera de los registros simples o dobles y la constante puede ser un dato de 8 o 16 bits. Este tipo de carga da lugar al llamado direccionamiento inmediato.

b) LD reg,reg

Esta instrucción de carga se utiliza para pasar datos de un registro a otro y ambos registros deben ser siempre de ocho bits, salvo en los casos antes señalados para registros dobles. El direccionamiento a que da lugar se denomina de registro y también implícito.

c) LD reg,(nn) y LD (nn),reg

Este modo de carga permite poder sacar datos de una dirección de memoria dada por (nn) o introducirlos en ella. Recuerde que una dirección viene dada en memoria por dos octetos de los cuales el primero es la parte baja y el segundo, la parte alta.

LD(nn),HL cargará en nn el contenido del registro L y en nn+1, el valor del registro H.

Este modo de carga sólo es posible utilizando registros dobles. El único caso permitido para registros simples es, como cabe esperar, el del acumulador. En este último caso:

LD A,(nn)

equivale a: A=PEEK(nn) y lo contrario a Poke(nn),A.

El direccionamiento a que da lugar este modo de carga se conoce como direccionamiento extendido.

d) LD reg,(rr) y LD (rr),reg

Este modo de carga es muy ingenioso y rápido ya que se aprovecha la posibilidad de que un registro doble apunte a una determinada dirección de memoria y el dato pueda ser leído directamente por un registro simple y a la inversa, es decir, meter un dato en la dirección de memoria a la que apunta cualquiera de los registros dobles.

El acumulador puede hacer esta "operación" con cualquiera de los registros dobles y los demás registros simples sólo pueden acceder a través de los pares de registros HL,IX e IY.

El modo de direccionamiento a que da lugar se conoce como indirecto a través de registro.

Veamos la resolución de un pequeño ejercicio:

Queremos que las direcciones 50000 y 50001 contengan los números 4 y 5 respectivamente utilizando el acumulador.

Utilizando este modo de direccionamiento tendríamos lo siguiente:

```
LD A,4
LD BC,50000
LD (BC),A
INC BC
INC A
LD (BC),A
```

Como verá, solamente utilizamos 9 octetos para codificar la operación en CM.:

3E 04 01 50 C3 02 03 3C 02

Instrucciones de intercambio

Realizan un canje entre sí de los valores de dos pares de registros. En BASIC existe una instrucción similar SWAP, que intercambian los valores de dos variables.

Los "canjes" posibles en CM. para el Z-80 son los siguientes:

```
EX DE,HL
EX AF,AF'
EX (SP),HL
EX (SP),IX
EX (SP),IY
```

Además, con la instrucción EXX se intercambian los dos juegos de registros de uso general, es decir:

BC con B'C'
DE con D'E'
HL con H'L'

Para un estudio más amplio sobre los registros alternativos nos remitimos al apartado de los registros del Z-80A.

Instrucciones aritméticas

El grupo de instrucciones aritméticas del Z-80 comprende las siguientes:

ADD sumar
ADC suma con acarreo
SUB restar
SBC restar con acarreo
INC incrementar en 1
DEC decrementar en 1

Además dispone de una resta ficticia que sólo afecta al registro F sin conservar el resultado: CP.

CP sirve para establecer una comparación con un número o bien con el contenido de un registro simple. También está permitida la comparación por direccionamiento indirecto al registro HL -CP (HL)- y a los registros IX e IY en cuyo caso hablaremos de direccionamiento indexado.

En todos los demás casos, las instrucciones aritméticas se realizarán a través del acumulador quedando afectados los indicadores "flags" correspondientes.

Mención aparte merecen las instrucciones INC y DEC que no modifican el acumulador (excepto INC A y DEC A, como es obvio).

Para las instrucciones aritméticas de 16 bits, HL realizará la misma función que el acumulador y sólo podremos sumar a través de él sin quedar afectados los "flags".

Instrucciones lógicas

Todas las instrucciones lógicas se realizan sobre el acumulador comparándose bit a bit cada uno de los ocho que componen un octeto.

Las operaciones lógicas permitidas son las siguientes:

AND
XOR
OR
CPL

Basándose en el Álgebra de Boole para estas operaciones y considerando que un bit puesto a 1 equivale a Verdadero y puesto a 0 equivale a Falso, se obtienen para cada operación lógica las siguientes tablas de verdad:

Operador lógico AND (A ∩ B)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

```
  01001101   A
  11000110   B
  --------
  01000100
```

Operador lógico OR (A ∪ B)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

```
  01001101   A
  11000110   (HL)
  --------
  11001111
```

Operador lógico XOR (A ⊕ B)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

```
  01001101   A
  11000110   (IX+d)
  --------
  10001011
```

Operador lógico NOT (CPL)

X	NOT X
1	0
0	1

```
01001101    A
--------
10110010    NOT A
```

Instrucciones de flujo de programa

Estas instrucciones son las equivalentes en CM al GOTO y a todas las instrucciones que modifican la secuencia lógica del programa según determinadas condiciones: IF THEN GOTO, ON GOTO, etc.

Con estas instrucciones formamos el esqueleto de un programa sobre las que después basar la estructura del mismo.

En este tipo de instrucciones, podemos considerar tres variantes:

a) Saltos relativos.

Tipifican un desplazamiento en complemento a dos. Esto es, +127 casillas de memoria hacia adelante y −128 hacia atrás a partir de la última posición del PC (Contador de Programa). Aunque no pueden acceder a la totalidad de las posiciones de memoria, son muy útiles y rápidas para saltos pequeños dentro del programa ya que su codificación sólo necesita dos octetos: uno para la instrucción y otro para el desplazamiento. Su nemónico es : JR DIS, donde DIS será la distancia positiva o negativa hacia donde debe continuar el programa.

Esta instrucción admite en su estructura dos formas:

Saltos condicionales:

Bifurcan dependiendo del valor de los distintos bits del registro F, de los que sólo puede utilizar el "flag" C de acarreo y el Z de resultado cero. Por tanto tendremos cuatro tipos de saltos condicionales relativos:

JR Z,DIS. Bifurca si el flag Z es igual a 1. Resultado cero.
JR NZ,DIS. Bifurca si el flag Z es igual a cero. Resultado no cero.
JR C,DIS. Bifurca si el flag C es igual a 1. Existe acarreo.
JR NC,DIS. Bifurca si el flag C es igual a cero. No existe acarreo.

Salto incondicional:

No depende del estado de ningún flag, y se posiciona directamente en la dirección hacia adelante o hacia atrás indicada. Son de la forma:

JR DIS.

b) Saltos directos.

Son instrucciones de tres octetos en las que el primero indica el código de operación y los dos siguientes el número de la dirección de memoria a la que debe dirigirse el PC.

Estas instrucciones son del tipo:

JP NN

y pueden dirigirse a cualquier zona de la memoria. Al igual que los anteriores, admite bifurcación condicional e incondicional. La incondicional no depende del estado de los flags, bifurcando directamente a la casilla señalada. Es el equivalente exacto en CM. a la instrucción GOTO número de línea, aunque aquí son posiciones de memoria en lugar de números de línea.

En cuanto a las condicionales, admiten las mismas bifurcaciones relativas a los flags C y Z (JP Z,nn; JP NZ,nn; JP C,nn y JP NC,nn); estando además permitidos nuevos condicionamientos de salto relativos a los flags P/V y S:

JP PO,nn. Bifurca si el flag P/V es igual a cero. Esto es, cuando existe paridad impar.

JP PE,nn. Bifurca si el flag P/V es uno. Paridad par.

JP P,nn. Bifurca si el flag S es igual a cero. Es decir, el resultado de una operación es de signo positivo.

JP M,nn. Bifurca si el flag S es igual a 1. O sea, el resultado de una operación tiene signo negativo.

c) Saltos indirectos.

Son unos tipos de saltos muy interesantes y económicos ya que la dirección de bifurcación no viene dada directamete sino que vendrá dada por el contenido de uno de los registros de 16 bits: HL,IX,IY

Este tipo de saltos son incondicionales y por tanto no admiten decisiones a través de los flags del registro F.

Tenemos los siguientes:

JP (HL)
JP (IX)
JP (IY)

Instrucciones de llamada a subrutinas

Estas instrucciones funcionan de manera análoga a las de BASIC GOSUB y RETURN, y al igual que ellas, deben ir emparejadas correctamente.

Su equivalente en CM. son:

Llamada = CALL nn
Retorno = RET o RET nn

Al igual que vimos para las instrucciones de salto directo (JP nn), también aquí es posible decidir la bifurcación según el estado de los distintos flags; tanto para la llamada como para el retorno.

Para la llamada utilizamos tres octetos donde el primero es el código de operación y los dos siguientes la dirección de memoria donde comienza la subrutina. Para el retorno se utiliza un sólo octeto.

Las instrucciones de llamada pueden ser incondicionales y condicionales. Las primeras son del tipo:

CALL nn

y en cuanto a las condicionales, como podemos bifurcar con los mismos condicionantes que vimos para los saltos directos tendremos los siguientes tipos de llamadas y retornos respectivamente:

Llamadas	Retornos
CALL NZ,nn	RET NZ
CALL Z,nn	RET Z
CALL NC,nn	RET NC
CALL C,nn	RET C
CALL PO,nn	RET PO
CALL PE,nn	RET PE
CALL P,nn	RET P
CALL M,nn	RET M

Llamadas a rutinas en página cero

Existe una instrucción especial de llamada para el Z-80 que utiliza direccionamiento en página cero. Este direccionamiento es llamado así porque la parte alta de las rutinas a las que accede es siempre cero.

La ventaja de este tipo especial de llamada RESTART, respecto a las anteriores es que solamente utilizan un octeto en lugar de tres, con lo que se ahorra tiempo y memoria.

Son del tipo: RST n, donde n sólo puede admitir uno de los valores siguientes:

RST &H 00	RST &H 20
RST &H 08	RST &H 28
RST &H 10	RST &H 30
RST &H 18	RST &H 38

De éstas podemos destacar

RST &H 00 que es la dirección del RESET

RST &H 18 que muestra el carácter contenido en el acumulador dentro del periférico en uso OUTDO

RST &H 20 que compara HL con DE restando en HL el valor de DE y según el resultado modifica los flags pero sin alterar ni HL ni DE.

Instrucciones de manejo de la pila

No nos detendremos aquí sobre este punto ya que lo veremos después dentro del estudio del microprocesador Z-80. Diremos, no obstante, que un adecuado uso de las instrucciones PUSH (escritura en la Pila) y POP (lectura de la Pila) permite realizar operaciones complejas con los registros ya que salvaguarda su valor momentáneamente para luego retomarlo y continuar con la operación en curso.

La salvaguarda de los registros se hace siempre por pares de 16 bits.

Un manejo inadecuado de la pila puede dar al traste con cualquier programa. Por tanto, para programas largos en CM. conviene calcular previamente la ubicación de la pila cargando el puntero (SP) mediante la instrucción:

LD SP,nn

donde nn será la dirección más alta de la zona de memoria reservada a la pila.

Cargador hexadecimal

```
10 CLEAR200,35500!
20 STOPON:ONSTOPGOSUB500
30 CLS:DEFSNG A-Z
40 KEY OFF:SCREENO:WIDTH37
50 POKE&HF3B1,4
60 FORI=35488!TO35499!
70 READA$
80 POKEI,VAL("&H"+A$)
90 NEXT:A$=""
100 DEFUSR=35488!:A=USR(0)
```

```
110 DATA 21,0,10,3e,20,1,0,30,cd,56,00,c9
120 LOCATE0,1:INPUT"DIRECCION DE COMIENZO";
B$:XX=VAL(B$)
130 N=1:D=209:V=0:S=0:VV=4256:P=49
140 X=VAL(B$):IF X<35500! OR X>63000!THEN 4
40
150 GOSUB320
160 LOCATE2,2:PRINTX;:INPUTA$
170 IF LEN(A$)>2 THEN 460
180 H=VAL("&H"+A$)
190 POKEX,H
200 S=S+H
210 C$=HEX$(PEEK(X)):IFLEN(C$)<2THENC$="0"+
C$
220 VPOKED,ASC(MID$(C$,1,1))
230 D=D+1
240 VPOKED,ASC(MID$(C$,2,1))
250 D=D+2
260 X=X+1
270 B$=STR$(X)
280 IFD>231+VTHEND=D+15:V=V+40:GOSUB320
290 LOCATE 0,2:PRINTSTRING$(30,32)
300 GOTO 160
310 END
320 '
330 IFD>960THENLOCATE0,1:PRINT"              ESP
ERE";SPC(15):BEEP:T=2
340 IFT>0THENFORI=0TO799:VPOKEVV+I,VPEEK(16
0+I):NEXT:IFT>1THENCLS:D=208:V=0:VV=VV+1024
:P=P+1:T=0ELSE510
350 VPOKE235,ASC("P")
360 VPOKE236,ASC(":")
370 VPOKE238,P
380 FORI=1TOLEN(B$)
390 VPOKED-7+I,ASC(MID$(B$,I,1))
400 NEXTI
410 IFN=1THENN=0:GOTO 430
420 D=D+1
430 RETURN
440 LOCATE0,1:PRINTSTRING$(30,32)
450 GOTO 120
460 LOCATE 30,2:BEEP:PRINT"ERROR"
470 FORI=0TO100:NEXT
480 LOCATE0,2:PRINTSTRING$(39,32)
```



```
490 GOTO 160
500 LOCATE3,2:PRINT" UN MOMENTO.";SPC(20)
509 T=1:GOTO 340
510 T=0:CLS:LOCATE0,0:PRINT"DIRECION INICIA
L";XX
520 PRINT"DIRECION FINAL";X
530 PRINT"SUMA DE LOS OCTETOS";S
540 INPUT"CORRECTO S/N ";A$
550 IFA$="S"ORA$="s"THENPOKE&HF3B1,24:SCREE
N0:KEYON:END
560 CLS:LOCATE0,0:PRINT"OBSERVE DATOS Y APU
NTE ERRORES"
570 PRINT"CURSOR DCHO. AVANZA PAGINA"
580 PRINT"CURSOR IZDO. RETROCEDE PAGINA"
590 PRINT"UTILICE < ESC > PARA SALIR ";:R=0
600 A$=INPUT$(1):A=ASC(A$)
610 IFA=28THENR=R+1:GOTO 650
620 IFA=29THENR=R-1:GOTO 650
630 IFA=27THENVDP(2)=0:CLS:S=0:GOSUB670:FOR
I=XXTOX:H=PEEK(I):S=S+H:NEXT:GOTO 510
640 GOTO 600
650 IFR+3<0THENR=0:GOTO 600ELSEVDP(2)=3+R
660 GOTO 600
670 INPUT"DIRECCION ERRONEA";E
680 INPUT"DATO ";B$
690 IFLEN(B$)>2THENBEEP:GOTO 680
700 POKEE,VAL("&H"+B$)
710 INPUT"MAS ERRORES S/N ";A$
720 IFA$="S"ORA$="s"THEN670ELSERETURN
```

Manejo del programa

Este programa es una aplicación muy personal de lo que podría ser un cargador hexadecimal para ubicar rutinas en código máquina.

El programa utiliza la memoria de video del modo de texto como archivo momentáneo de la rutina introducida desde el teclado.

Una vez ejecutado, el programa pide la dirección inicial donde se quiere ubicar la rutina, rechazando las direcciones menores de 35500 y mayores de 63000, aunque esta última puede ser cambiada a voluntad del lector con la variable X de la línea 140.

A continuación vaya usted introduciendo los datos de su rutina que irán colocándose sucesivamente en su posición de memoria correspondiente.

Observe como hemos dividido la pantalla en dos partes: la de arriba donde se teclean los datos y la inferior donde se tiene una visión global de los datos que han sido introducidos, agrupados en columnas de ocho. Es ésta la parte que posteriormente será memorizada en la VRAM para un posible revisionado en caso de haber introducido un dato erróneo.

Si la rutina es larga y sobrepasa la capacidad de la página, el programa generará automáticamente una nueva página archivando la anterior.

Cuando haya finalizado de teclear los datos, pulse [CTRL][STOP] y compruebe si la suma de los octetos es correcta. Si es así responda afirmativamente a la pregunta. En caso negativo siga las instrucciones que el programa le indique.

8. Solución a los ejercicios de la página 9

1. Codificación en Hexadecimal:

63456 = &HF7E0
10021 = &H2725
23441 = &H5B91
20034 = &H4E42
12876 = &H324C
465 = &H01D1

Mediante el ordenador lo averiguaremos con:

?HEX$(X)

2. Codificación en hexadecimal de números binarios:

E2 ; 5D BC

Lo averiguamos por medio de:

?HEX$(&B Num)

3. Codificación decimal de números binarios:

230 132 19

Se averigua con:

?&B Núm.

4. Codificación binaria de números hexadecimales:

0100110111001100 1111101000010110
1010110011011100 1011101000011111
1110100110101111

Se averigua con:

?Bin$(&H núm.)



Capítulo 2

Configuración interna del MSX

El microprocesador Z-80A

El famoso Z-80 surgió como un microprocesador de ocho bits que acumulaba todos los perfeccionamientos del momento. La versión mejorada, conocida como Z-80A, aún poseía otros aditamentos importantes.

El Z-80 sale al mercado hacia el año 1976, desarrollado por ZILOG. Anteriormente, unos dos años atrás, se había desarrollado el famoso microprocesador 8080, el primero de ocho bits con éxito comercial.

Las siglas Z-80 corresponden, por tanto: Z a ZILOG y 80 porque es un descendiente del 8080 al que imita y supera.

Sobre un chip cuya superficie es alrededor de un 20% mayor que la del 8080, el Z-80 incorpora cerca de 8000 transistores (más del doble que el 8080), y posee 158 instrucciones que se amplían a 696 con los diferentes modos de direccionamiento.

Curiosamente, el desarrollo de los microprocesadores de 8 bits no ha progresado posteriormente debido a que los fabricantes se dedicaron prioritariamente a potenciar nuevos microprocesadores de 16 y 32 bits. Por esto, y por su avanzada concepción, el Z-80 sigue en cartel en el campo de los de ocho bits. No es de extrañar, por tanto, que haya sido elegido por la norma MSX para su primera generación de ordenadores. Y seguramente su hermano mayor, el Z-800 de 16 bits, será el microprocesador que incorporen los MSX en un futuro próximo, cuando sea comprobado exhaustivamente. Pero no nos adelantemos a los acontecimientos y veamos resumidamente quién es este personaje: el Z-80A.

Características

La capacidad de direccionamiento del Z-80A, es decir, la cantidad de memoria que puede manejar en todo momento, es de 64 Kbytes.

En efecto: siendo un microprocesador de ocho bits, tenemos que 8^2=64K. Para el Z-800 serían 16^2=8^2*4=256K, siendo fácil de obtener rápidamente la máxima capacidad de direccionamiento para microprocesadores superiores.

La frecuencia del reloj es de 2.5MHz y cuatro MHz para el Z-80A. En los MSX, se ha sincronizado a 3.58 MHz, para no llevarlo a su límite de trabajo.

La alimentación es sólo de 5 voltios.
Los modos de direccionamiento son:

- Implícito
- Inmediato
- Relativo
- Directo
- A través de registro
- Indexado

Los registros, incluidos tanto los de 8 bits como los de 16 son 22. Algunos de los 16 bits se pueden desdoblar en dos de ocho.
El bus de direcciones es de 16 bits y el de datos de 8.

Funcionamiento

El corazón del Z-80 es su reloj interno y sus "latidos" proporcionan el ritmo de cálculo y ejecución de las instrucciones que le llegan de la memoria donde han de estar contenidas previamente. Es por esto por lo que cuanto más alta sea la frecuencia del reloj de un microprocesador, en general, más rápido será en su ejecución.
Ya en el interior del Z-80, dos son los bloques encargados de interpretar y procesar las instrucciones que le son suministradas:

- El decodificador de instrucciones
- La ALU (Aritmetic Logic Unit) o Unidad Aritmética y Lógica.

El decodificador interpreta cada byte que le llega por el Bus de datos y activa, en consecuencia, los diferentes circuitos dependiendo de la programación que haya introducido el fabricante en este bloque. Como el decodificador es de 8 bits, sólo son posibles $2^8=256$ instrucciones. Pero como se ha querido disponer de 696 posibilidades de interpretación, se amplió a dos bytes el código de algunas instrucciones. Es el caso de aquellas cuyo primer byte en hexadecimal es: CB, DD, ED y FD.

Ejemplo:

LD A, 5D (5D es el dato) significa carga (LoaD) el acumulador A con &H5D. Su codificador en memoria es: 3E 5D. 3E es el código que se corresponde con la orden: "carga el registro A con el dato que viene a continuación". 5D es, por tanto, el dato.
La codificación de dos bytes sería:
LD A,(IX+04) que significa: "Carga A con el dato que apunta el registro IX, cuya dirección está dada por el contenido de IX + el byte que le sigue, conocido como desplazamiento". Esta instrucción sería en CM. (abreviatura de Código Máquina) FD 7E 04. Donde FD y 7E son los dos bytes del código y 04 el desplazamiento.

La ALU es la encargada de sumar, restar o hacer operaciones lógicas entre dos bytes y entregar el resultado por el registro A o HL, afectando al registro de estado o Flag, denominado F.
Todos estos procesos, como ya hemos visto, son controlados por la "lógica interna" accionada por el decodificador de instrucciones, bajo la acción del reloj que encadena con sus "pulsaciones" todos los acontecimientos.
En la ejecución de un programa tenemos otros dos registros especializados:

- El PC (Program Counter), o contador del programa
- Registro F (Flag), o registro de estado.

El registro PC

El PC es un registro especial de 16 bits, por tanto, puede acceder a todas las direcciones de memoria, que son 65536.
Como sabemos, un Kbyte equivale exactamente a 2^{10} = 1024 bits. 64K serán, por tanto 1024*64 = 65536. Es decir, puede controlar desde 0 hasta 65535 posiciones o casillas de memoria (&H0000–&HFFFF, en hexadecimal).
El PC lleva la dirección de memoria de cada uno de los bytes de la instrucción en curso y se incrementa de uno en uno cada vez que el decodificador de instrucciones devuelve de la memoria los bytes que va interpretando. Al final de cada instrucción se ve aumentado tantos bytes como esta contenga.
Cuando se hace un Reset, todos los registros se ponen a cero incluido el PC. Por ello empieza en la dirección &H0000 de la ROM apareciendo el conocido mensaje de:

MSX system
Versión 1.0
Copyright 1983 by Microsoft.

El registro F

El registro F, de estado o Flag, indica en todo momento el resultado operativo del microprocesador; lo cual sirve al programa para tomar decisones y efectuar saltos condicionados, etc.
El Z-80 se comunica en todo momento con la memoria y periféricos mediante tres buses de señales:

Bus de direcciones
Bus de datos
Bus de control

El bus de direcciones presenta hacia el exterior la dirección del byte en proceso o la dirección del periférico en funcionamiento. Está compuesto por 16 bits que se numeran desde &HA0 hasta &HA15.

El bus de datos está compuesto de 8 bits desde &HD0 hasta &HD7. Por el pasan datos en hexadecimal para leer la memoria o para escribir en ella, según las directrices del programa.

El bus de control es específico de cada microprocesador. Para el Z-80 este bus se puede desglosar en cuatro conceptos:

El control de buses
El control de los procesos de lectura y escritura
El control de las señales de temporización
El control del sistema

En cuanto a los registros, además del PC y del F ya mencionados brevemente, existen dos bancos intercambiables de uso general de seis registros cada uno, un registro acumulador, dos registros índice de 16 bits, un puntero de pila y dos registros de 8 bits para el control de refresco y vector de interrupciones.

Los registros del Z-80

Un registro puede ser considerado como un circuito especial dentro del microprocesador destinado al almacenamiento temporal de información digital, es decir, en codificación binaria. Su capacidad de almacenamiento es el equivalente a una palabra del procesador; y en nuestro caso, al ser un procesador de ocho bits, la longitud de la palabra será equivalente a ocho bits, es decir, lo que conocemos como byte. Si nuestro procesador fuera de 16 bits, la longitud de la palabra sería de dos bytes.

Quizá esta definición no nos aclare mucho. Por eso vamos a dar una definición desde el punto de vista de su funcionamiento, para comprender mejor su lógica interna.

Vamos, pués, a considerar los registros como si fueran simples celdillas, casillas o posiciones de memoria especiales directamente accesibles por el microprocesador sin tener que pasar a través de una dirección.

Estas celdillas "especiales" no tienen, por tanto, dirección y se distinguen entre sí sólo por su nombre.

Ejemplo:

Si queremos realizar la operación X con el contenido de la celdilla de memoria de dirección dada por &H×2×1, (siendo ×2 la parte alta y ×1 la parte baja) se convierte gracias a estos registros en: "haz la operación X con el contenido del registro A", por ejemplo.

Así pués, el uso de los registros es imprescindible para toda la operatividad interna del microprocesador, tanto para la comunicación con el exterior como para el almacenamiento temporal de la información. Esta última cualidad será la más importante y de la cual se sirva al programador para llevar a cabo la tarea propuesta.

Los registros del Z-80A

	Grupo 1		Grupo 2	
DE USO GENERAL	B	C	B′	C′
	D	E	D′	E′
	H	L	H′	L′
ACUMULADOR	A		A′	
INDICADORES	F		F′	
PUNTERO DE PILA	SP			
CONTADOR PROGRAMA	PC			
INDICE	IX			
	IY			
INTERRUPCIÓN Y REFRESCO	I		R	

1. Los registros de uso general

El grupo 2 es denominado grupo de registros alternativo (ARS o Alternate Register Set).

Cada uno de estos registros puede ser usado individualmente, como registros de ocho bits, o combinados para formar una pareja de 16 bits. Estas parejas no pueden formarse arbitrariamente sino que ya están prefijadas de la siguiente manera:

HL,BC,DE más sus respectivos pares alternativos H′L′,B′C′,D′E′.

La utilización individual o por pares dependerá del criterio del programador, utilizando las instrucciones apropiadas.

Así, por ejemplo, el registro DE puede ser usado como D, o como E o como DE. Esta última configuración posibilita alcanzar directamente cualquier dirección de la memoria y operar con valores hasta 65535 (&HFFFF). Cuando usemos esta última posibilidad, el registro D llevará la parte alta y el C la parte baja. Esto será fácil de recordar si tenemos presente el caso del registro HL (H=high, alto, y L=low, bajo), y por extensión aplicable a cualquier par de registros.

Como el microprocesador no puede acceder directamente a la totalidad de estas registros, será el programador el que decida dentro del conjunto HL,H'L' etc. con qué pareja va a trabajar si con la HL o la H'L'. Lo normal es trabajar con la HL, BC o DE que son por así decir la cara activa del conjunto.

Para fijar mejor el concepto vamos a considerar el símil de una ficha de dominó:

Cuando usted coge una ficha de dominó mira, por así decir, la cara activa pero enseña hacia afuera la cara en negro. Considere la cara que usted ve como una pareja de registros con su parte alta y su parte baja, por ejemplo el cinco doble. Considere también que la cara en negro, por arte de encantamiento pueda almacenar otro valor. Si usted gira la ficha podrá acceder a esa información oculta, y a la vez, ocultar la que antes sólo conocía. Este "encantamiento" se puede realizar con la instrucción EXX que intercambia BC con B'C', DE con D'E' y HL con H'L'.

2. El Acumulador y el Flag

En cuanto a una clasificación estricta, podrían haber sido introducidos en el apartado anterior como otra pareja más de registros de la forma AF,A'F'; pero debido a sus características tan distintas a las otras parejas de registros, hemos preferido tratarlos con más detalle en este apartado.

Lo primero que debemos considerar es que no podemos reunirlos para formar un registro doble de 16 bits, sino que deben funcionar siempre aparte como dos de ocho. Esto que podría parecer extraño en principio, no lo resultará tanto si vemos que el registro A es un acumulador (de ahí el nombre), por el que pasan todos los resultados operativos de 8 bits y al que están referidas la mayoría de las instrucciones.

El registro F, hermano en la sombra del registro A, es el que lleva el estado del microprocesador en todo momento. Cada uno de sus ocho bits tiene un cometido especial, que veremos dentro de un momento.

El acumulador es el único registro de ocho bits al que se le puede sumar directamente un número.

Ejemplo:

Queremos sumar a un registro simple, el D, un dato, el número 13. La instrucción ADD D,13 que sería la vía directa no es posible realizarla. Sin embargo utilizando el acumulador la tarea propuesta quedaría de esta manera:

LD A,D. "Carga el acumulador con D"
ADD A,13. "Carga A con el valor 13"
LD D,A. "Transfiere a D la suma efectuada en A"

El resultado ha sido igual que el mandato inexistente: ADD D,13.

Esto puede parecer un dispendio casi absurdo pero no lo es tanto si vemos su codificación hexadecimal en CM.

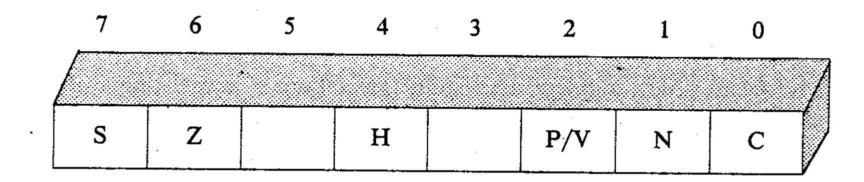
7A C6 D 57

Como vemos, sólo hemos necesitado 4 bytes teniendo un sólo registro dedicado que es el A. Se comprenderá fácilmente la importancia del acumulador para todo tipo de operaciones.

E registro F es el encargado de vigilar los resultados contenidos en A y decidir, según nuestras necesidades hacia donde debe bifurcarse el programa etc.

Cuando empiece a utilizar el CM. notará la potencia de este registro, difícil de modificar directamente ya que está concebido para que sea el propio Z-80 el que lo utilice al ejecutar las instrucciones lógicas, aritméticas, etc.

Veamos ahora cada uno de los ocho bits de estado que componen este registro.



De los ocho bits significativos hay dos (el 3 y el 5) que no tienen significado para el programador, pero si para el microprocesador ya que se basa en ellos para la ejecución de ciertas operaciones internas. Los seis bits restantes son utilizados por el microprocesador de la siguiente manera:

Bit 0 (C):Acarreo (Carry).

Este bit es el indicador de acarreo. Es puesto a 1 por el Z-80 cuando el resultado de operar un dato rebosa en el acumulador el máximo valor posible de ocho bits, es decir, 255 o &HFF en hexadecimal (que corresponde a 2^8); en caso contrario, permanece a 0. Este bit se puede poner a 1 directamente mediante la instrucción especial 'SCF'. Las expresiones lógicas 'AND', 'OR', 'XOR' ponen este bit a 0.

El bit de acarreo queda también afectado por numerosos tipos de desplazamiento y rotación debidos a las instrucciones 'RLA', 'RLCA', 'RRA', etc.

Si se operan datos de 16 bits por mediación de los registros dobles, entonces el bit C se pone a 1 cuando el resultado de una operación excede de 65535 o &HFFFF ((2^8)*(2^8)=2^16). Este bit también actua como bit de acarreo en instrucciones de sustracción 'SBC'.

Bit 1 (N):Suma/resta en BCD (Add/substract).

Este bit es afectado en operaciones realizadas en BCD, esto es, Decimal Codificado en Binario donde necesariamente están prohibidos los números hexadecimales desde la A hasta la F. El "flag" N se pone a 1 en la operación de resta en BCD y a 0 en las operaciones de suma lógica, rotación y desplazamiento.

Bit 2 (P/V):Paridad/Desbordamiento (Parity/Overflow).

Este bit es utilizado como indicador de paridad o como bit de desbordamiento, según las instrucciones.

Para las instrucciones aritméticas sirve de indicador de desbordamiento de capacidad. El desbordamiento se produce cuando el resultado de una operación aritmética excede el valor comprendido entre −128 (&H80) y +127 (&H7F) para operaciones de ocho bits. De modo similar, se producirá desbordamiento entre −32768 (&H8000) y +32767 (&H7FFF) en operaciones de 16 bits.

El flag N es utilizado como bit de paridad en las instrucciones lógicas, de rotación y desplazamiento. Estas lo ponen a 1 si en el resultado, la suma de bits en estado lógico 1 es par y a 0 si es impar.

Bit 4 (H):Medio acarreo (Half carry).

Este bit sólo se utiliza en operaciones en BCD en las que el acarreo que se produce cada cuatro bits es determinante para su operatividad en sumas y restas.

Bit 6 (Z):Resultado Cero (Zero).

Este bit es alterado por las instrucciones aritméticas y lógicas y también por la de comprobación de bit.

En general, éste indicador se pone a 1 si el resultado de la última operación es cero y a 0 cuando el resultado es distinto de cero. Aquí también se apoyan las instrucciones de salto que efectuan o no una bifurcación dependiendo del resultado entregado a este bit por la instrucción inmediatamente anterior.

Bit 7 (S):Signo (Sign).

Este flag indica el signo del octeto contenido en el acumulador.

En este bit queda reflejado el bit de mayor peso de una operación. De esta forma serán considerados positivos los números comprendidos entre 0 y 127 (&H7F) porque su bit número 7 es un cero; y negativos desde 128 (&H80) hasta 255 (&HFF) porque su bit número siete es un 1.

En las operaciones cuyo resultado implica 16 bits, el flag S se pone a 1 entre &H8000 y &HFFFF, es decir, cuando el bit de mayor peso, que ahora será el 15, es 1 y son positivos entre &H0000 y &H7FFF, porque el bit más significativo es 0.

Es importante tener presente que el registro de flags va unido al acumulador cuando ambos se utilizan como un registro de 16 bits. Nos servirá de utilidad a la hora

de almacenar temporalmente su contenido para recuperarlo más tarde tras la ejecución de una parte del programa.

3. El Registro SP

Este registro es llamado "Puntero de Pila" (Stack Pointer). Contiene una información de 16 bits que representa una dirección de memoria. Es el encargado de la gestión de la pila.

Una Pila, también llamada LIFO (Last In, First Out) puede ser considerada como una serie de celdillas que ocupan posiciones contiguas de memoria cuyo acceso es gestionado por el registro que nos ocupa.

Para utilizar una imagen plástica, podríamos interpretar una pila como el montón de platos apilados en los estantes de la cocina de un hotel. Cada dato a escribir en la pila sería el equivalente a un plato que iremos poniendo encima del montón, de tal forma que el último plato colocado sería el primero en estar disponible. Pues bien, poner un plato en la Pila equivale a la operación de escritura y retirar el plato superior supone la operación de lectura.

De la misma manera, en un ordenador se pueden ir guardando una serie de octetos en una tabla de memoria llamada normalmente "Cola".

La gestión de la Pila por el registro SP se realiza de la manera siguiente:

Al comenzar, cuando la Pila está vacía, el registro SP contiene en dos octetos (para poder acceder a cualquier posición de memoria) la dirección de una casilla de memoria que puede ser inicializada por nosotros mediante la instrucción:

```
LD SP,nn
```

donde nn será la dirección elegida como inicio de la Pila.

Cuando escribimos en la Pila el registro SP es decrementado en 1, escribiendose a continuación el dato en la celda de memoria cuya dirección viene dada por el registro. Los datos de la Pila se irán colocando, así, en direcciones contiguas de forma decreciente. Cuando leemos se recorrerá el camino inverso.

Es importante que al comienzo del programa reservemos una zona de memoria para la Pila inicializando SP con la dirección superior de la zona de memoria elegida.

Las instrucciones de que dispone el programador para trabajar con la Pila se pueden resumir dos: PUSH para entrada y POP para salida, combinadas ambas con los registros dobles.

Ejemplos:

PUSH nn introducirá el contenido del par de registros indicado por nn en la pila apuntada por SP. Esta instrucción ejecuta los siguientes pasos: decrementa el valor del registro SP y carga el octeto de orden superior del par de registros indicado por nn en la dirección especificada por SP; acontinuación vuelve a decrementar el registro SP y carga el octeto de orden inferior.

POP nn introduce en el par de registros indicado por nn los dos primeros octetos de la Pila apuntada por SP. Esta instrucción ejecuta los siguientes pasos: carga en la parte inferior del par de registros indicado por nn el octeto de la dirección especificada por el registro SP; incrementa el registro SP y carga el siguiente octeto en la parte superior del par de registros, después vuelve a incrementar el registro SP.

4. Registros de Interrupción y Refresco (I y R)

Estos dos registros son utilizados por algunas configuraciones "hardware" y corresponden a la Unidad Central (CU).

El registro I, de ocho bits, contiene la parte superior de la dirección a partir de la cual ha de realizarse una bifurcación si ha habido una interrupción de programa; en este caso la parte inferior del registro representa el tipo de interrupción dado por la unidad central.

El registro R es utilizado como contador para "refrescar" a intervalos regulares el contenido de la RAM dinámica. Se impide que se pierdan las informaciones memorizadas mediante la contínua recarga del mismo contenido de memoria.

5 Registros Índice

Además de los registros de uso general ya vistos: HL, DE, BC y AF, existen otros dos registros dobles de 16 bits que trataremos aquí, debido a que se utilizan en una técnica especial denominada "direccionamiento indexado". Estos registros se denominan IX e IY.

La particularidad de su uso estriba en que están pensados para poder acceder a datos de la memoria considerados como tablas, por el programador. Están especialmente indicados para todo el software de gestión de cadenas y archivos etc. Es por esto que requieren un byte más que los otros registros, encargado de llevar el "desplazamiento" que será añadido a la dirección apuntada por IX o IY; de tal manera que la dirección absoluta vendrá dada por el valor del registro mas el del byte de desplazamiento.

EL desplazamiento puede ser hacia adelante o hacia atrás según el signo del bit 7 de este byte (−128 a +127 en complemento a dos como máximo).

Por ejemplo:

```
LD A,(IX+d)
```

Se cargará en el acumulador el valor de la casilla de memoria a la que apunte el registro IX incrementada en 127 o decrementada en −128 según el valor del desplazamiento.

Aunque pueden ser utilizados indistintamente por el programador tanto el IX como el IY; nosotros le aconsejamos que utilice el IX, ya que si utiliza frecuentes llamadas a la ROM el IY será utilizado por el sistema para apuntar a determinadas variables.

La configuración del MSX

En este apartado vamos a dar una visión somera de las interioridades de un ordenador MSX ya que esto ayudará a una mejor comprensión de su funcionamiento, estando por tanto más capacitados para hacernos "entender" por el mismo.

Ante todo hay que decir que frente a otros ordenadores los MSX tienen "Chips" específicos encargados de determinadas funciones además de la CPU o centro de control. En otros ordenadores suele ser la CPU la que realiza todas las funciones, además de la suya propia, como pueden ser procesador de video, generador de sonido, etc. En los MSX estas funciones se reparten entre diversos "miembros" siendo el Z80 el que controla todo el proceso como si de un organismo perfectamente organizado se tratara.

Podemos distinguir como partes fundamentales de su aparato las siguientes:

Microprocesador Z80-A de ocho Bits
VDP (Video Display Processor) de Texas Instruments IC TMS-9918
PSG (Programmable Sound Generator) AY3-8910 de General Instruments
PPI (Programmable Peripheral Interface) IC PPI 8255 de Intel
ROM 32K con intérprete Basic MSX de Microsoft

Sobre el Z80-A no insistiremos más puesto que ya hemos hablado extensamente de él.

El VDP lo iremos conociendo gradualmente a lo largo de todo el libro, ya que es la parte fundamental sobre la que gira la obra. Con él se manipulan pantallas, sprites, colores etc. Aquí sólamente diremos que posee 16k de memoria totalmente independiente de la principal, numeradas de 0 a 16383. Estas casillas pueden ser repartidas en bloques a nuestro gusto variando en los diversos modos de pantalla su contenido y distribución.

El PSG abarca un campo totalmente distinto al VDP pero no por ello menos importante ya que es el encargado de generar el sonido en cualquier forma que nosotros queramos pero siempre en los límites de su programación.

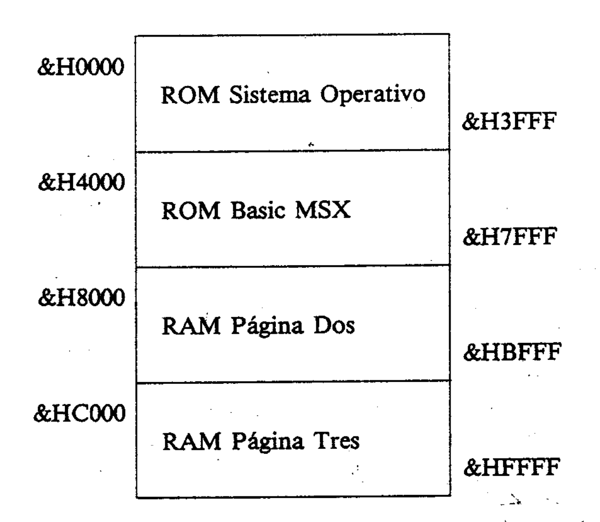
Explicar detalladamente como se programa, desborda nuestra pretensión, por lo tanto, expondremos aquí su estructura interna:

Tres Canales A, B, C
Un Generador de Ruido
Un Mezclador
Un Controlador de Amplitud
Un Generador de Envolvente
Tres Convertidores Digital/Analógico
Ports de Entrada/Salida

Una de las características principales de este circuito radica en que una vez programado deja libre a la CPU para otras tareas y por tanto la producción de efectos sonoros relativamente largos, no influirá en la velocidad de ejecución del programa.
El PPI es como el "relaciones exteriores" de la CPU. En efecto, él será el encargado de controlar los distintos periféricos, considerando como tales el teclado, el cassette o incluso cómo está paginada la memoria.

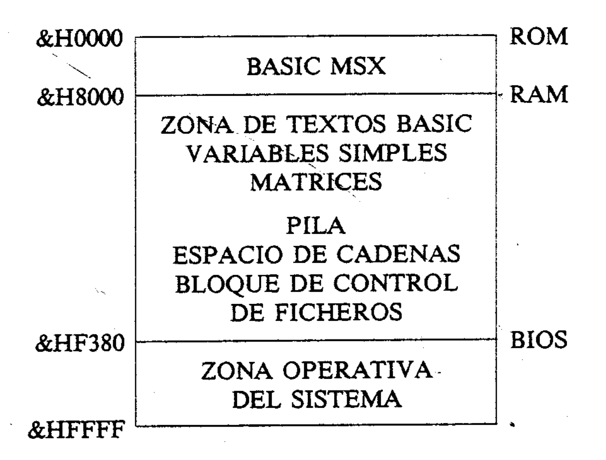
Estructuración de la memoria

Como ya sabemos, el Z80 es un procesador de ocho bits cuya máxima capacidad de direccionamiento son 64k. Consideremos un ordenador MSX de 64k, por ser la configuración más usual. A efectos de trabajo, esa memoria estaría subdivida en cuatro bloques o páginas de 16k cada una con lo que en teoría poseeríamos 65536 casillas de memoria libres a las que la CPU podría acceder. Sin embargo sólo podríamos programar en CM y de manera harto difícil.
Al conectar el ordenador, la memoria, aún siendo la misma, es distribuida de una manera distinta para facilitar el manejo por el usuario. De esta manera las dos páginas inferiores están ocupadas por dos páginas de ROM: la inferior contiene los 16k del sistema operativo y la inmediatamente superior los otros 16k del intérprete Basic. Las dos páginas restantes quedan libres para el usuario como RAM fácilmente accesible.
El esquema de la memoria podría quedar como sigue:



Sin embargo los 32K teóricamente libres para el usuario nos quedarán aun más reducidos, en cerca de 3.5K debido a que son necesarios como región de comunicación del sistema operativo. Esto que en principio podría parecer una desventaja, no lo es tanto ya que pueden servir para un cierto control del sistema por parte del usuario.

Así pués la zona libre real queda reducida aproximadamente a unos 29K que van desde la dirección &H8000 a &HF380.
La cantidad de bytes libres que quedan en definitiva, ha suscitado muchas críticas al standard MSX comparándolos con otros ordenadores. Sin embargo, en un sentido estricto, se debe responder que se poseen 16K adicionales en VRAM. Esto es muy importante si tenemos en cuenta que en otros ordenadores se debe utilizar parte de la memoria RAM como archivo de pantalla, mientras que en MSX, esta parte es totalmente independiente.
Centrándonos, pués, en las 29568 posiciones de memoria restantes podemos distribuirlas de acuerdo al siguiente esquema:



A la vista del esquema anterior, observamos cómo a partir de la posición de memoria &H8000 se empiezan a archivar en formato empaquetado los programas Basic. Esto será válido tanto para ordenadores de 32K como de 64K RAM, ya que a efectos de programación Basic el resultado será tener sólamente disponibles las dos páginas de 16K superiores. Si su ordenador fuera de 16K, entonces sólo estaría disponible la página superior, comenzando a archivarse los listados Basic a partir de la dirección &HC000.
En esta casilla inicial ya sea &H8000 o bien &HC000 deberá existir un cero, ya que en caso contrario, el intérprete entendería que es un programa en código máquina y no un listado Basic lo que vendría a partir de ella. En las dos casillas siguientes tendremos archivada la dirección de memoria donde comienza a almacenarse el siguiente número de línea.
Todas las líneas Basic finalizan con un Byte 0 dando a continuación mediante dos Bytes más la dirección de memoria donde comienza la línea siguiente. De esta manera se van encadenando las sucesivas líneas hasta llegar al final del texto Basic. El final del texto será identificado mediante tres bytes cero.
A partir de aquí, el intérprete memoriza su longitud en la variable RAM del BIOS &HF6C2-C3 en la formal habitual de Byte bajo, Byte alto. En ese momento y a partir de ahí se memorizarán las variables; empezando por las simples y después las

de matriz. Posteriormente reservará la dirección de la pila, dejando la cola de la memoria disponible para el área de cadenas y posible zona de manejo de ficheros si los hubiera.

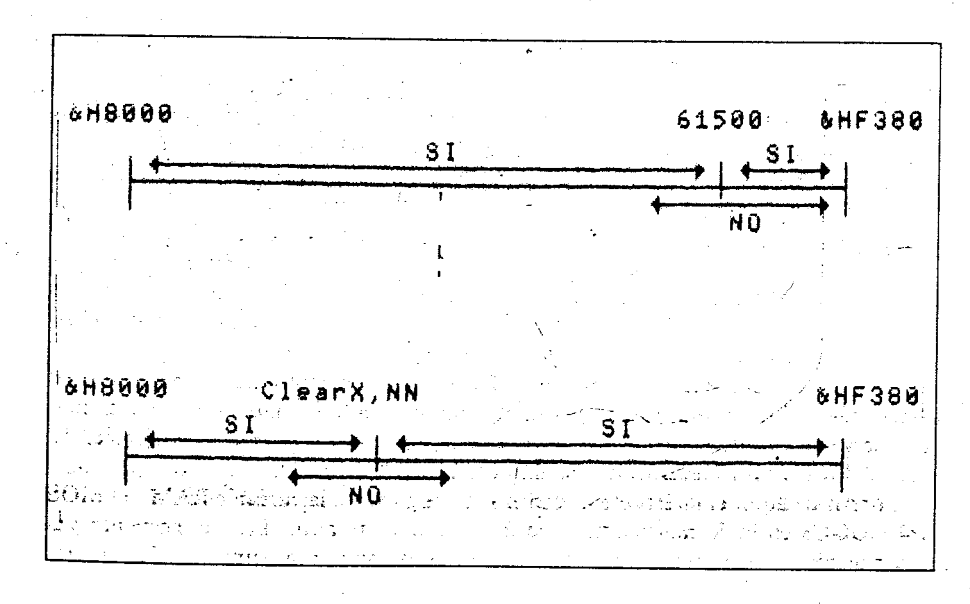
Una buena costumbre de programación es utilizar la sentencia CLEAR para reservar un posible espacio de memoria utilizable por CM. En este caso se podrá acceder a la totalidad de la memoria hasta la dirección &HF380. A partir de aquí, e ineludiblemente comienza la región de comunicación que posteriormente veremos.

Otra de las normas a tener en cuenta será la posible carga en cassette de estas 29568 casillas. En efecto, al encender el ordenador y sin variar mediante CLEAR la Ramtop, sabremos que el límite de memoria corresponde a &HF380. Esta dirección vendrá indicada además en la variable RAM &HFC4A y &HFC4B. Compruebelo mediante el siguiente mandato:

```
?Hex$(peek(&HFC4A)+256*peek(&HFC4B))
```

En este caso, el ordenador reserva como zona de fichero desde la casilla 61500 aproximadamente a la 62336. Por eso si intentáramos cargar mediante BLOAD"cas:" un programa en CM que fuera desde 61100 a 62300 atravesando el semáforo de la casilla 61500 probablemente se produciría un "cuelgue" o un Reset del ordenador. En cambio podremos cargar o salvar en cinta la parte que vaya de 61500 en adelante o hacia atrás siempre que su extensión no atraviese por el límite de la Ramtop.

Si mediante un Clear X,NN reservamos un espacio de cadenas X, estableciendo en NN la nueva Ramtop, la restricción anterior será la misma relativa a NN. En este caso NN equivaldría a 61500 (Ramtop de inicialización) y no se podría cargar mediante Bload"cas:" un programa que atraviese por NN.



La razón de todo esto es debida a la propia rutina de la ROM encargada de gestionar la instrucción BLOAD, ya que tiene en cuenta la Ramtop antes de retornar al editor del Basic.

Tenga en cuenta estos detalles cuando quiera aprovechar al máximo la memoria libre para el usuario.

La RAM del sistema

Si usted es un poco curioso, habrá observado que en ocasiones encontramos efectos inesperados "pokeando" en determinadas direcciones valores determinados. Esto se debe a que los programadores que crearon el sistema operativo del MSX, pensaron en la posibilidad de modificar el sistema operativo según las necesidades del usuario. Para ello reservaron una zona de memoria modificable (RAM) desde la cual manejar el flujo de las principales rutinas (ROM) del sistema que, por supuesto no son directamente modificables. A esta zona se le suele llamar región de comunicación y podemos dividirla en dos grandes zonas:

- La Ram del sistema
- La tabla de vectores

La Ram del sistema es una zona en la que se guardan los principales parámetros y variables internas. Contiene principalmente elementos de un octeto (variables internas y semáforos) y elementos de dos octetos (variables internas y direcciones). Comienza en &Hf380 y termina en &Hfd99.

A continuación pasamos a exponer los principales parámetros indicando para su comodidad el nombre, la dirección, la longitud y la función de las principales variables.

NOMBRE	DIR.	LG.	FUNCIÓN
ARG	F847	016	Acumulador Secundario
ARYTAB	F6C4	002	Dirección de la tabla de variables de matriz
ASPECT	F931	002	Zona de trabajo Circle
ATRBYT	F3F2	001	Octeto atributo
AUTFLG	F6AA	001	Indicador de AUTO si = 0
AUTINC	F6AD	002	Incremento entre líneas para AUTO
AUTLIN	F6AB	002	Número actual de línea utilizado por AUTO
BAKCOL	F3EA	001	Color de fondo
BASETB	F91F	010	Valores tablas del VDP
BASLOD	FCAE	001	Indicador carga BASIC
BASROM	FBB1	001	Si BASIC está en Rom = 0

NOMBRE	DIR.	LG.	FUNCIÓN
BDRCOL	F3EB	001	Color de borde
BOTTOM	FC48	002	Dir, comienzo de la Ram
BRDATR	FCB2	001	Color de borde en Paint
BUF	F55E	258	Zona Buffer del teclado
CAPST	FCAB	001	Indicador de Caps Look
CASATR	F3FC	020	Gestión del cassette
CLIKSW	F3DB	001	Indicador click. Si = 0
CLMLST	F3B2	001	Núm. caracteres para TAB
CLPRIM	F38C	014	Rutina de salto al interior de slot en banco 0
CNSDFG	F3DE	001	Flag teclas de función 0 = no visibles.
CODSAV	FBCC	001	Código del cursor
CONSAV	F668	001	Archivo temporal del código de instrucción.
CONTXT	F666	002	Dir. carácter actual txt
CRTCNT	F3B1	001	Núm. líneas por pantalla
CSRSW	FCA9	001	Cursor on / Cursor off
CSRX	F3DD	001	Posición horizontal actual del cursor.
CSRY	F3DC	001	Posición vertical actual del cursor.
CSTYLE	FCAA	001	Carácter del cursor.
CURLIN	FA1C	002	Núm. actual de la línea en curso de ejecución.
DAC	F776	016	Acumulador Matemático.
DATLIN	F6A3	002	Indicador de la última línea DATA leida
DATPTR	F6C8	002	Puntero octeto siguiente al último carácter en modo de ejecución.
DEFTBL	F6CA	026	Tabla de declaración de Variables.
DIMFLG	F662	001	Indicador para DIM.
DORES	F664	001	Tipo de Operador.
DOT	F6B5	002	Núm. actual de línea.
DRWANG	FCBD	001	Ángulo para DRAW.
DRWFLG	FCBB	001	Indicador para DRAW.
DRWSCL	FCBC	001	Escala para DRAW.
DSCTMF	F698	002	Dirección siguiente octeto disponible en la tabla de cadenas.
ENDBUF	F660	001	Fin de Buffer.
ENDFOR	F6A1	002	Puntero instrucción FOR.
ENDPRG	F40F	005	Puntero para RESUME-NEXT
ENSTOP	FBB0	001	Posible recalentamiento si es igual a cero.
ERRFLG	F414	001	Núm. del último error.
ERRLIN	F6B3	002	Núm. línea con error
ERRTXT	F6B7	002	Puntero para RESUME.
ESCCNT	FCA7	001	Contador Sentencia SCAPE

NOMBRE	DIR.	LG.	FUNCIÓN
EXPTBL	FCC1	004	Comienzo zona de trabajo en conmutación cartucho.
FBUFFR	F7C5	43	Zona de trabajo BCD.
FLBMEM	FCAE	001	Indicador de carga para programa BASIC
FLGINP	F6A6	001	Flag para INPUT-READ.
FNKFLG	FBCE	010	Flag para ON KEY GOSUB.
FNKSTR	F87F	160	Contenido teclas función
FORCLR	F3E9	001	Color texto. (Utilizado por sentencia COLOR)
FRCNEW	F3F5	001	Contiene &Hff.
FRETOP	F69B	002	Dirección máxima del espacio de cadenas.
FSTPOS	FBCA	002	Primera posición de carácter en INLIN.
FUNACT	F7BA	002	Uso temporal para recogida de basura.
GETPUT	F3FA	002	Dir, octeto actual a leer en buffer de teclado
GRPACX	FCB7	002	Acumulador gráfico X
GRPACY	FCB9	002	Acumulador gráfico Y.
GRPATR	F3CD	002	Dir. tabla de atributos de Sprites.
GRPCGP	F3CB	002	Dir. tabla del generador de patrones en Screen 2.
GRPCOL	F3C9	002	Dir. tabla de colores en Screen 2.
GRPHED	FCA6	001	Encabezamiento de càrácter gráfico.
GRPNAM	F3C7	002	Tabla del nombre de patrones en Screen 2.
GRPPAT	F3CF	002	Tabla del generador de sprites en Screen 2.
GXPOS	FCB3	002	Posición horizontal del cursor en gráficos.
GYPOS	FCB5	002	Posición vertical del cursor en gráficos.
HIMEM	FC4A	002	Dir. final memoria RAM.
INSFLG	FCA8	001	Indicador modo inserción
INTCNT	FCA2	002	Contador de Intervalo.
INTFLG	FC9B	001	Indicador de Interrupción
INTVAL	FCA0	002	Valor de intervalo en ON INTERVAL GOSUB
JIFFY	FC9E	002	Jiffy.
KBUF	F41F	318	Buffer para la codificación de una línea BASIC.
KEYBUF	FBF0	040	Buffer codificación de tecla
LINL32	F3AF	001	Long. línea en Screen 1.
LINL40	F3AE	001	Long. línea en Screen 0.
LINLEN	F3B0	001	Long. línea actual.
LINTTB	FBB2	024	Tabla de terminadores de línea
LINWRK	FC18	040	Operaciones de proceso de pantalla
LOHMSK	F949	001	Zona trabajo para PAINT.

NOMBRE	DIR.	LG.	FUNCIÓN
LOWLIM	FCA4	001	Lectura de Cassette.
LPTPOS	F415	001	Posición cabeza impresora
MAXFIL	F85F		Comienzo zona de parámetros para la manipulación de ficheros.
MCLTAB	F956	002	Zona de trabajo para PLAY.
MEMSIZ	F672	002	Valor superior de la memoria utilizable por el BASIC. (Modificable con CLEAR).
MLTATR	F3D7	002	Dir. tabla de atributos de sprites en Screen 3.
MLTCGP	F3D5	002	Dir. tabla del generador de patrones en Screen 3.
MLTCOL	FSD3	002	Dir. tabla de colores en Screen 3.
MLTNAM	F3D1	002	Dir. tabla del nombre de patrones en Screen 3.
MLTPAT	F3D9	002	Dir. tabla del generador de sprites en Screen 3.
NTMSXP	F417	001	Tipo de impresora. MSX=0
OLDKEY	FBDA	011	Antiguo estado de tecla.
OLDLIN	F6BE	002	Núm. de línea establecido por STOP y END.
OLDSCR	FCB0	001	Antiguo modo de pantalla
OLDTXT	F6C0	002	Dir. del último octeto ejecutado.
ONEFLG	F6BB	001	Vale &Hff durante tratamiento ON ERROR.
ONELIN	F6B9	002	Núm. línea de tratamiento de error.
ONGSBF	FBD8	001	Flag de condición para ON ...GOSUB.
PADX	FC9D	001	Valor de X para el pad.
PADY	FC9C	001	Valor de Y para el pad.
PARM1	F6E8	100	Tabla de parámetros para funciones def. por usuario
PARM2	F750	100	Direcciones de parámetros
PRTFLG	F416	001	Flag de impresora. 1=impresora; 0=pantalla.
PTRFLG	F6A9	001	Indicador de modo programa o modo directo.
PUTPNT	F3F8	002	Buffer teclado (put).
QUEUES	F3F3	002	Dir. tabla de espera.
RDPRIM	F380	011	Lectura slots banco cero.
RG0SAV	F3DF	008	Contenido de los ocho registros del VDP. (0-7).
RTYCNT	FC9A	001	Control de interrupción.
RUNBNF	FCBE	001	Flag entrada/salida bin.

NOMBRE	DIR.	LG.	FUNCIÓN
SAVENT	FCBF	002	Comienzo de BSAVE.
SAVSTK	F6B1	002	Guarda la dirección del SP para manipular error.
SAVTXT	F6AF	002	Puntero para RESUME.
SCNCNT	F3F6	001	Sincronización de exploración de tecla.
SCRMOD	FCAF	001	Modo actual de pantalla.
STKTOP	F674	002	Dirección superior del puntero de pila (SP)
STREND	F6C6	002	Dir. comienzo de espacio disponible de memoria.
SUBFLG	F6A5	001	Indicador para FOR y USR
SWPTMP	F7BC	008	Almacenamiento temporal para SWAP.
T32ATR	F3C3	002	Dir. tabla de atributos de sprites en screen 1.
T32CGP	F3C1	002	Dir. tabla del generador de patrones en screen 1.
T32COL	F3BF	002	Tabla de colores screen 1.
T32NAM	F3BD	002	Dir. tabla del nombre de patrones en screen 1.
T32PAT	F3C5	002	Dir. tabla del generador de sprites en screen 1.
TRCFLG	F7C4	001	TROFF=0, TRON=1
TXTATR	F3B9	002	Dir. tabla de atributos de sprites. screen 0.
TXTCGP	F3B7	002	Dir. tabla del generador de patrones. screen 0.
TXTCOL	F3B5	002	Dir. tabla de colores en screen 0.
TXTNAM	F3B3	002	Dir. tabla del nombre de patrones en screen 0.
TXTPAT	F3BB	002	Dir. tabla del generador de sprites. screen 0.
TXTTAB	F676	002	Dir. comienzo de texto en programa BASIC.
USRTAB	F39A	020	Tabla de las direcciones definidas por la instrucción DEFUSR n=.
VALTYP	F663	001	Tipo de variable presente en DAC.
VARTAB	F6C2	002	Dirección de la tabla de variables simples.
VLZADR	F419	002	Utilizada por VAL.
VOICAQ	F975	128	Comienzo de las tres direcciones de las listas de espera musicales.
WRPRIM	F385	007	Rutina de escritura de los slots del banco 0.

LA TABLA DE VECTORES

La otra parte del área de parcheo, o región de comunicación es la llamada Tabla de Vectores (Hook). Comienza en la &HFD9A y termina en &HFFC9.

En esta zona podemos, como luego veremos, interceptar las grandes rutinas de la ROM del Sistema. Esto es posible porque cada llamada a una rutina Basic pasa por uno de los vectores situados en esta zona por medio de una instrucción CALL.

Cada vector está compuesto de cinco octetos cargados inicialmente con el código de retorno RET (&H09) y por lo tanto, a no ser que esté interferido por las direcciones utilizadas por el Disc Basic, volverá directamente a la ROM sin hacer nada.

Para interceptar una rutina deberemos escribir en el grupo correspondiente una instrucción de salto incondicional JP (HC3) y la dirección a la cual transferir la rutina teniendo en cuenta la parte alta y baja de la dirección convenida.

A continuación pasamos a detallar una lista con las direcciones, el nombre y la función de dichos vectores.

NOMBRE	DIR.	FUNCIÓN
ATTR	FE1C	Al inicio de la rutina de atributos.
BAKU	FEAD	En la rutina de vuelta atrás.
BINL	FE76	Al final de la instrucción SAVE.
BINS	FE71	En la rutina de instrucción SAVE.
BUFLIN	FF8E	En la rutina de la instrucción LIST al convertir el código en clave (Buffer Linea).
CHGET	FDC2	Al inicio de lectura de un carácter.
CHPUT	FD44	Al inicio de escritura en pantalla del carácter contenido en A en modo de Texto.
CHRGET	FF48	Al inicio de la recogida de un carácter.
CLEARC	FED0	En la rutina de inicialización de la tabla de variables.
CMD	FE0D	Al inicio de la instrucción CMD
COMPRT	FF57	En el tratamiento de la instrucción PRINT.
COPY	FE08	Al inicio de instrucción COPY.
CRDO	FEE9	En la rutina de impresión de un carácter seguido de retorno de carro.
CRUNCH	FF20	En la rutina de conversión de una línea Basic en código.
CRUSH	FF25	Al inicio de búsqueda de una instrucción en la tabla.
CVD	FE49	Al inicio de instrucción CVD (conversión a doble precisión).
CVI	FE3F	Al inicio de instrucción CVI (conversión a entero).
CVS	FE44	Al inicio de instrucción CVS (conversión simple precisión.

NOMBRE	DIR.	FUNCIÓN
DEVN	FEC1	En la rutina nombre de unidad
DGET	FE80	En la rutina lectura del Disco. En la rutina de ejecución en modo directo.
DOGRPH	FEF3	Al inicio funciones gráficas.
DSKC	FEEE	En la rutina de lectura de un carácter de disco.
DSKF	FE12	Al inicio de instrucción DSKF (espacio libre en disco)
DSKI	FE17	Al inicio instrucción de escritura en disco.
DSKO	FDEF	Al inicio instrucción de lectura de disco.
DSPC	FDA9	Al inicio rutina de actualización del cursor.
DSPFNK	FDB3	Al inicio de la rutina de visualización contenido de las teclas de función.
EOF	FEA3	Al inicio de la función EOF.
ERAC	FDAE	Al inicio rutina de borrado del cursor.
ERAFNK	FDB8	Al inicio de la rutina borrado del contenido teclas función.
ERRF	FF02	Al final impresión del mensaje de error.
ERRO	FFB1	Al inicio de tratamiento error.
ERRP	FEFD	Al inicio impresión del mensaje de error.
EVAL	FF70	Al inicio de evaluación de una expresión.
FIEL	FE2B	Al inicio instrucción FIELD.
FILES	FE7B	Al inicio instrucción FILES.
FILOUT	FE85	Al inicio de salida a fichero.
FINEND	FF1B	Al final de la interpretación.
FING	FF7A	Al final evaluación funciones.
FINI	FF16	Al final de la interpretación de una instruccción.
FINP	FF5C	Al final tratamiento de la instrucción PRINT.
FORM	FFAC	Utilizado por el Call Bios en &H0148 en rutina formateadora del disco.
FPOS	FEA8	Al inicio función FPOS.
FREEUP	FF9D	Antes de la búsqueda de un espacio libre para una cadena de caracteres.
FRMEVL	FF66	Al inicio de evaluación de una fórmula.
FRQL	FF93	Al inicio instrucción POKE.
GENDSP	FEC6	En la rutina general de reparto de unidades.
GETPTR	FE4E	Al inicio de lectura del puntero de fichero.
GONE′	FF43	Al inicio instrucciones de flujo de programa.
INDS	FE8A	Al inicio comprobación rutinas.
INIP	FDC7	A la inicialización del VDP.
INLIN	FDE5	Al inicio de INPUT (lectura de línea).

NOMBRE	DIR.	FUNCIÓN
IPL	FEA3	Al inicio de la instrucción IPL (carga inicial del programa).
ISFL	FEDF	Al inicio de comprobación existencia de fichero E/S.
ISMI	FF7F	Al inicio instrucción MID$.
ISREW	FF2A	Al inicio descubrimiento de palabra reservada en fase CRUNCH.
KEYCOD	FDCC	Al inicio lectura de teclado.
KEYI	FD9A	Llamada en &G004b. Tratamiento de interrupciones. VDP.
KILL	FDFE	Al inicio instrucción KILL.
KEYEAS	FDD1	Llamada en &H0F10 Antes de convertir el carácter enviado por el teclado según la tabla situada en &H1003.
LIST	FF89	En la rutina de (L)LIST.
LOC	FE99	Al inicio de la función LOC.
LOF	FE9E	En la función LOF (long. fichero).
LPTO	FFB6	En la rutina de escritura por impresora.
LPTS	FFBB	En la rutina de Test de estado de la impresora. (CHKPTR).
LSET	FE21	En la instrucción LSET.
MAIN	FF0C	Llamada entrada del intérprete.
MERGE	FE67	En la instrucción MERGE.
MKD$	FE3A	En la función MKG$. Creación de doble precisión.
MKI$	FE30	En la función MKI$. Creación de entero.
MKS$	FE35	Al inicio de instrucción MKS$. Creación de simple precisión.
NAME	FDF9	En el comienzo de cambio nombre.
NMI	FDD6	Tratamiento de interrupción no enmascarable.
NODEV	FEB7	Si falta nombre de unidad.
NOFOR	FE58	Falta FOR en la instruc. OPEN.
NTFL	FE62	En la instrucción CLOSE.
NTFN	FF2F	Cuando la palabra reservada va seguida de número de línea.
NULOPE	FE5D	En la rutina de apertura de fichero nulo.
ONGO	FDEA	En la instrucción ON GOTO y ON GOSUB.
OUTDO	FEE4	En la salida de carácter por pantalla o impresora.
PARDEV	FEB2	Al comienzo de análisis de nombre de unidad.
PINLIN	FDDB	En la rutina línea de programa.
PLAY	FFC5	En la instrucción PLAY.
PRGET	FEF8	Al final ejecución de programa.
QINLIN	FDE0	En la rutina de impresión de ? en la instrucción INPUT.

NOMBRE	DIR.	FUNCIÓN
READY	FF07	En impresión OK.
RETURN	FF4D	En la instrucción RETURN.
RSET	FE26	En instrucción RSET.
RSLF	FE8F	En instrucción INPUT$.
RUNC	FECB	En la instrucción NEW o RUN.
SAVD	FE94	Para grabar en la unidad actual.
SAVED	FE6C	Al comienzo de SAVE.
SCNEX	FF98	En la conversión de número de línea en puntero y a la inversa.
SCREEN	FFC0	Al principio instrucción SCREEN.
SETFIL	FE53	En la rutina de inicialización de puntero sobre un fichero.
SET	FDF4	En la instrucción SET.
STKERR	FEDA	En rutina de error de la pila.
TIMI	FD9F	Llamada en &H0C53. En la rutina de lectura del registro de estado del VDP. Tratamiento de interrupción.
TOTE	FDBD	En rutina de retorno a modo de texto desde un modo gráfico.
TRMNOK	FF61	En el tratamiento de un DATA o INPUT incorrectos.
WIDTH	FF84	En la instrucción WIDTH.

EJEMPLOS

En este último apartado, le proponemos algunos trucos utilizando la región de comunicación. Los dos primeros funcionan alterando directamente el valor de las Variables del Sistema y los otros dos "parcheando" los vectores.

Limitación de página sin borrado automático

De todos es sabido que al ejecutar un Width, se nos produce un borrado automático de la pantalla. A veces es necesario alterar tanto la anchura de línea como el número de líneas por pantalla.

Utilice las direcciones &HF3AE o &HF3AF para limitar el ancho de pantalla (WIDTH) para screen 0 o screen 1 respectivamente. Y &HF3B1 para limitar el número de líneas por pantalla.

Pruebe a dar diferentes valores a las casillas señaladas y compruebe los efectos con el cursor. También puede tener presente la casilla &HF3DE que se encarga del flag de las teclas de función.

Reserva de memoria por debajo de la tabla de texto

Supongamos que queremos reservar 5K de memoria para programas en CM y que además, para despistar, queremos situarlos por debajo de la zona donde se almacena el texto Basic de un programa.

```
POKE &KF676,1
POKE &HF677,&H94
POKE &H9400,0
```

Con esto logramos que la tabla de instrucciones del programa se situe en la casilla 37888 (&H9400). La única condición es que en esa primera casilla introduzcamos un cero.

Supresión y modificación del mensaje OK

A veces es necesario que no aparezca en pantalla el mensaje OK, como por ejemplo en el programa de utilidad Gestor de Pantalla; pero también podemos modificarlo:

```
10 CLEAR 200,56000!
20 POKE &HFF08,&HC0
30 POKE &HFF09,&HDA
40 FORI=56000! TO 56022!: READ A$
50 POKE I,VAL("&H"+A$):NEXT
60 POKE &HFF07,&HC3
70 DATA CD,23,73,21,C9,DA,C3,31,41,0A,0D
80 DATA 51,55,45,20,48,41,47,4F,3F,0D,0A,00
```

Al ejecutar el programa lograremos sustituir el mensaje OK por el mensaje QUE HAGO?

El programa símplemente utiliza el Vector READY para desviar el curso normal de la rutina a la casilla 56000 donde comienza una pequeña subrutina que pone el mensaje y ejecuta, para continuar, un salto a la dirección &H3141.

Protección contra listados:

A veces es bueno proteger los programas de las miradas indiscretas. Para ello basta con interceptar el Vector LIST y escribir la dirección del RESET (Restart &H0). Con lo cual el posible mirón se quedará sin programa.

```
POKE &HFF08,0
POKE &HFF09,0
POKE &HFF07,&HC3
```


Capítulo 3

Modos de pantalla

1. El procesador de video

Descripción:

Sin duda, la pantalla es el modo más común y directo que el ordenador posee para ponerse en contacto con el programador. Para realizar esta operación, los ordenadores de la norma MSX disponen de un procesador especial llamado VDP (Video Display Processor) o procesador de video basado en un circuito complejo de Texas Instruments, el TMS 9918A, que está dotado de memoria propia y que se encarga de dirigir todas las señales necesarias para que una imagen sea proyectada en pantalla.

Al decir que está dotado de memoria propia, nos daremos cuenta fácilmente de que esto significa que no ocupa ningún lugar sobre el bus del procesador central que tratará al VDP como si en realidad fuera un periférico comunicándose con él por medio de dos "ports" de entrada/salida (&H99 y &H98). Esta configuración supone para nosotros una ventaja adicional: disponer de más espacio para la codificación de programas.

Básicamente, el VDP posee cuatro modos de funcionamiento que se corresponden con los diferentes modos de pantalla:

- Modo de texto
- Modo gráfico I
- Modo gráfico II
- Modo multicolor

La memoria de video

La memoria de video o VRAM (Video Random Access Memory) es la parte de su ordenador MSX destinada en exclusiva a la gestión de pantalla. Todo lo que usted escriba en ésta memoria tendrá su correspondiente efecto en pantalla y todo lo que escriba directamente en la pantalla ocupará un lugar físico en la memoria.

La memoria de video dispone de 16384 casillas de lectura/escritura numeradas desde la 0 hasta la 16383 (&H0-&H3FFF).

Cuando usted activa en su ordenador cualquiera de los modos de pantalla mediante la instrucción BASIC: **SCREEN**, la memoria de video se inicializa señalando diversas zonas de almacenamiento llamadas tablas. Cada tabla posee una dirección base determinada por el sistema a la que usted puede acceder mediante la instrucción Base.

Para cada modo de pantalla disponemos de cinco parámetros que especifican la preparación de la VRAM. Estos parámetros son los siguientes:

0 Tabla de la zona de pantalla
1 Tabla de colores
2 Tabla de Generador de caracteres
3 Tabla de atributos de figuras móviles
4 Tabla del generador de Figuras móviles

La tabla de la zona de pantalla

En modo de texto es un bloque de 40*24=960 casillas contiguas de memoria que en nuestro ordenador comienza en la dirección &H00 y acaba en &H03BF (0-959)

En modo gráfico (I-II) comienza en &H1800 y su extensión es de 32filas * 24 columnas = 768 casillas consecutivas de 8 octetos.

En modo multicolor el bloque comienza en &H0800 y su extensión es análoga a los modos gráficos I y II.

Tabla de colores

Es utilizada en los modos gráficos I y II. La dirección de comienzo para ambas es &H2000 (8192) y su extensión depende de la opción elegida.

- En modo de texto aprenderemos a definir los colores basándonos en el registro 7 del VDP.
- En modo multicolor usaremos la tabla del generador de patrones para definir los colores de cada cuadrado de 4*4 puntos que componen cada bloque.

La tabla del generador de patrones

En modo de texto esta tabla contiene los datos de cada uno de los 256 caracteres que conocemos. Estos caracteres pueden ser programados a nuestro gusto, como veremos. La dirección de comienzo se encuentra en &H800 (2048) y la extensión equivale a los 256 caracteres multiplicados por 8 octetos que definen cada uno de ellos.

En los modos gráficos I y II, la tabla comienza en &H00 y su extensión depende de la configuración especial de cada opción: 256*8 octetos para modo grafico I y (256*3)*8 octetos para el modo gráfico II.

Tabla del generador de figuras móviles

Esta tabla contiene los datos necesarios para el diseño de los famosos "sprites" y se combina con el registro 1 del VDP para ofrecer en pantalla el tamaño final que hayamos elegido. La dirección de comienzo para todos los modos excepto el de texto, que no los usa, se sitúa en &H3800 (14336). La extensión (2048 octetos) vendrá dada de operar 256 "sprites", máximo número posible, por lo 8 octetos que definen cada figura. Si bien sólo podremos portar 1 por cada plano (32).

Tabla de atributos de las figuras móviles

Esta tabla contiene la información necesaria para definir los colores de las figuras que hayamos diseñado, su colocación en la pantalla y el número identificativo de la figura. Para esta tarea emplea cuatro octetos para cada "sprite", que multiplicados por los 32 planos de que disponemos, supone una extensión de 128 octetos situados a partir de la dirección &H1B00 (6912) para todos los modos de pantalla que permiten figuras móviles.

Los registros del VDP

El VDP dispone de ocho registros de lectura/escritura y otro adicional de solo lectura o de estado numerados del cero al ocho. Estos registros definen los diferentes parámetros y las direcciones base para los distintos bloques de la VRAM.

Podemos acceder y modificar los valores de estos registros directamente con la instrucción especial VDP(n)=X, donde n será el número de registro y X el valor a introducir; o también Print VDP(n), para leer. No obstante, para controlar por separado cada bit del registro, lo cual le aconsejamos, usaremos la instrucción:

```
A$=BIN$(VDP(n)):?string$(8-len(A$),"0")+A$
```

Registro 0. VDP(0)

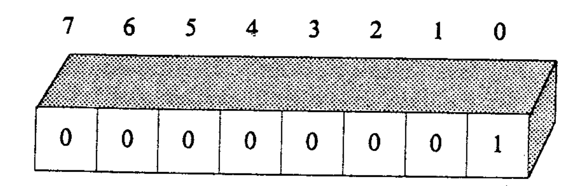
Los valores iniciales para cada modo de pantalla son los siguientes:

VDP (0)	Screen 0	Screen 1	Screen 2	Screen 3
V. inicial	000	000	002	000

De este registro pueden interesarnos los bits cero y uno. El bit cero sirve para aceptar una videoseñal externa. Normalmente no se usa y por tanto está a cero. Su puesta lógica a uno significa la aceptación de videoseñal.

Podemos probar a encender este bit introduciendo el mandato:

VDP(0)=&B00000001



Acabamos de perder la señal de color. No se preocupe. Pruebe ahora a dar diversos colores con el mandato BASIC que usted conoce, Color tinta, fondo, borde. Observará que las tonalidades cambian ofreciendo gamas de grises desde el blanco al negro. En esto se basa la técnica reciente de colorear películas antiguas por ordenador. El ordenador se encarga de realizar el proceso inverso: identificar las tonalidades de grises del "master" y codificarlas según una tabla de colores.

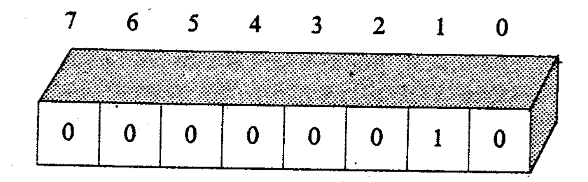
Siempre que el bit cero está en estado lógico 1 (encendido) obtendrá este efecto.

Restauremos el valor inicial.

El bit uno sirve para situarnos en modo gráfico 2, si lo ponemos a uno. Está a cero en los demás casos.

Compruébelo usted mismo introduciendo el siguiente mandato:

VDP(0)=&B00000010



Todos los demás bits deben estar a cero.

No obstante puede teclear las siguientes líneas y sacar sus propias conclusiones:

```
10 FORI=0T0255
20 VDP(0)=I
30 FORJ=1T0500
40 NEXTJ, I
50 END
```

Como usted comprobará, el hecho de recorrer toda la tabla de valores del VDP(0), se reduce a tres posibilidades: Videoseñal externa conectada, modo gráfico dos, o las dos posibilidades a la vez.

Registro 1. VDP(1)

Los valores iniciales para cada modo de pantalla son los siguientes:

VDP (1)	Screen 0	Screen 1	Screen 2	Screen 3
V. inicial	240	224	224	232

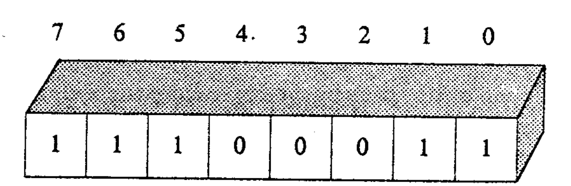
De este registro nos interesa estudiar la función de casi todos sus bits.

Cuando desde BASIC usted utiliza el segundo parámetro de la sentencia: SCREEN,n; define tamaño y ampliación de las figuras móviles mediante un valor comprendido entre 0 y 3. Pues bién, los bits cero y uno de este registro son los encargados de guardar el valor de n que usted ha entregado.

El bit cero es el encargado de ampliar o no las figuras móviles. Cuando este bit esté a 0, no habrá ampliación y cuando esté a 1 ampliará una figura independientemente de su tamaño. De esto último se encarga el bit uno: si está "encendido", puesto a 1, "entenderá" que la figura es de 16×16, y si está a 0, su tamaño será de 8×8. Al encender el ordenador ambos están inicializados a cero.

Teclee el siguiente mandato:

VDP(1)=&B11100011



Con la instrucción anterior el registro ha quedado modificado de la siguiente manera: bit cero y uno encendidos. Lo cual quiere decir que tendremos definidos los posibles "sprites" de 16×16 puntos y además los veremos ampliados al doble (32×32 puntos). Esto equivale a la instrucción BASIC: Screen, 3.

Por tanto, combinando estos dos bits tendremos las siguientes posibilidades:

Screen,0	0	0	Figura móvil de 8×8 sin ampliar
Screen,1	0	1	Figura móvil de 8×8 ampliada
Screen,2	1	0	Figura móvil de 16×16 sin ampliar
Screen,3	1	1	Figura móvil de 16×16 ampliada

Por favor, teclee este programa.

```
10 SCREEN1
20 VDP(1)=&B11100000 ' valor inicial en scr
een1
30 FORI=0TO31
40 VPOKE14336+I,VPEEK(0+65*8+I)
50 NEXT
60 PUTSPRITE0,(125,90),1,0
70 END
```

Verá usted una "A" en el centro de la pantalla. Como ya se habrá dado cuenta, es un "sprite". Lo que quizá no sepa es que el "sprite" es de 16×16 y diseñado como:

A C
B D

Como una imagen vale más que mil palabras, teclee usted en mandato directo:

VDP(1)=&B11100001
VDP(1)=&B11100010
VDP(1)=&B11100011

Aunque a usted todo esto le parezca mucho rodeo porque el mandato screen con sus diferentes parámetros lo resuelve fácilmente, nosotros insistimos en ello porque es así como realmente funciona su ordenador.

Aunque en el capítulo correspondiente a los "sprites" le explicaremos detenidamente el porqué de lo que vamos a hacer, sirvan ahora como adelanto los siguientes mandatos, para comprender mejor cómo trabaja el procesador de video.

Suponiendo que seguimos teniendo el "sprite" en el centro de la pantalla, teclearemos ahora:

VPOKE 6912,8

Con este mandato situamos en la celdilla de memoria correspondiente al plano cero de nuestra figura móvil un valor que es interpretado por el procesador como la coordenada vertical donde debe situar nuestro "sprite".

VPOKE 6913,20

Con este otro mandato igual al anterior salvo el incremento en uno de la casilla anterior, fijamos la nueva coordenada horizontal hacia donde debe desplazarse el "sprite".

VPOKE 6915,15

Con esto decimos al procesador que nos cambie el color negro que tenía por el blanco, cuyo código es el 15.

Con esto, queremos llamar la atención del lector acerca del proceso de "razonamiento" de su MSX. El "pobre" sólo entiende de números y de casillas donde almacenarlos. Nuestra tarea como programadores es hacer que estos números cobren forma, por así decir, según nuestros deseos o nuestras necesidades.

En nuestro caso, nos hemos limitado a poner tres números en el sitio preciso, para conseguir lo que pretendíamos.

Aunque continuamos en BASIC, nos acercamos casi al lenguaje directamente comprensible por el ordenador. La instrucción Put Sprite se reduce simplemente a introducir cuatro números en cuatro celdillas de memoria. Sólo nos falta un pequeño peldaño para entrar en CM.

Sigamos ahora con la descripción de los restantes bits que componen este registro.

El bit dos debe estar siempre a cero. Aunque usted pueda alterar su valor, no se lo aconsejamos.

Los bits tres y cuatro indican al ordenador, por el método de exclusión si está o no en modo gráfico. La misión la realizan de la siguiente manera:

Bit tres "encendido" significa que el ordenador se posiciona en modo multicolor. Si está a cero indica cualquiera de los demás casos.

Bit cuatro "encendido" activa el modo de texto, y "apagado", cualquiera de los restantes modos.

Screen,0	0	0	Modo de Escritura
Screen,1	0	0	Modo Gráfico I
Screen,2	0	0	Modo Gráfico II
Screen,3	0	1	Modo multicolor

El bit cinco está destinado al control de interrupciones. Cuando encendemos el ordenador, lo encontraremos a 1; lo cual quiere decir que las interrupciones de tipo "on Sprite, on interval, etc." son autorizadas por el sistema. Si cambiamos el bit a cero, las interrupciones no serán permitidas.

Este bit puede servir muy bien al propósito de proteger un programa BASIC teniendo en cuenta que no debemos utilizar las sentencias de tipo On ... Gosub que utilizan este bit para bifurcar el flujo del programa.

Así, por ejemplo, si queremos que el sistema detecte la colisión de dos "sprites"; utilizaremos en lugar de On Sprite Gosub número de línea, las sentencias IF ... THEN para conseguir el mismo resultado.

Veamos un ejemplo:

```
10 KEYOFF
20 SCREEN1,0
30 COLOR1,7,7
40 VDP(1)=&B11000000 ' DESCONECTAMOS LAS IN
TERRUPCIONES
50 FORI=0TO7:VPOKE14336+I,VPEEK(0+65*8+I):N
EXT
60 FORI=0TO7:VPOKE14344+I,VPEEK(0+90*8+I):N
EXT
70 X=0:Y=250
80 PUTSPRITE0,(X,100),1,0
90 PUTSPRITE1,(Y,100),15,1
100 X=X+1:Y=Y-1:BEEP
110 IFX=Y THEN VDP(1)=&B11100000:GOTO 130
120 GOTO 80
130 PLAY"s1m500o3ao4ao5ao6a"
140 IF PLAY(0) THEN 140 ELSE 40
```

Si ejecuta este programa, notará la imposibilidad de interrumpirlo. No se asuste. Mantenga presionadas ⟨CTRL+STOP⟩ hasta que el programa permita la interrupción.

Este sistema puede ser muy útil para cargar un programa con cabecera. En la cabecera desconectará usted las interrupciones impidiendo que alguien "curiosee" lo que haya pensado.

Ahora vamos a realizar esto mismo en CM.

Para traducir la instrucción: VDP(1)=&B11000000, deberemos reducirla a los siguientes pasos:

Cargar un registro con el valor 1, que es el núm. de VDP que vamos a modificar. A continuación cargar otro registro con el valor &B11000000, que es el dato a introducir y acudir después a una rutina de la ROM que nos resuelve el trabajo "pesado".

En primer lugar hallamos el equivalente en hexadecimal del núm. &B11000000. Para ello nos servimos del mandato siguiente:

Print Hex$(&B11000000)

El ordenador nos ofrecerá el resultado "C0". Ahora estaremos en condiciones de escribir la rutina:

- Cargar el registro B con &HC0 — LD B,&HC0
- Cargar el registro C con &H01 — LD C,&H01
- Acudir a la dirección de la ROM &H0047 — CALL &H0047
- Retornar a Basic — RET

Codificada en Hexadecimal quedaría de la siguiente manera:

06 C0 0E 01 CD 47 00 C9

Cargue los datos por ejemplo en la dirección 50000 y acuda con la instrucción defusr=50000:a=usr(0)

Si todo ha ido bien, deberá quedar desconectadas todas las interrupciones y al no estar en modo de programa no le quedará más remedio que hacer uso del Reset.

Siempre que quiera escribir desde máquina en un registro del VDP utilizando esta llamada a la ROM (&H0047), deberá cargar en el registro B el contenido y en el C el número del VDP que desee modificar.

La ROM posee muchas rutinas que facilitan la programación en CM. A lo largo de los siguientes capítulos le iremos mostrando algunas de las más utilizadas.

Veamos ahora algún ejemplo de utilización de los bits tres y cuatro.

```
10 ' PASO AL MODO MULTICOLOR POR MEDIO DEL
BIT TRES.
20 KEY OFF
30 SCREEN1
40 FORI=0TO23:PRINT"VAMOS A COLOREAR LA PAN
TALLA":NEXT
50 FORI=1TO1500:NEXT ' RETARDO
60 COLOR1,1,1
70 VDP(1)=&B11101000 ' PASAMOS A MODO MULTI
COLOR
80 FORI=1TO1500:NEXT ' RETARDO
90 VDP(1)=&B11100000 ' REINICIALIZAMOS
100 COLOR 15,4,4
110 END
```

El bit seis del registro es el encargado de encender o apagar la pantalla. La pantalla será encendida si el bit está a 1, y apagada si está a 0.

Normalmente siempre está a uno, pero podemos aprovecharlo para diversos fines, por ejemplo crear un dibujo en modo gráfico II con la pantalla apagada y después, que aparezca de repente.

Ejemplo:

```
10 ON INTERVAL=300 GOSUB 440:INTERVALON
20 DEFSTR S
30 COLOR 15,4,4:SCREEN2
40 ' VDP(1)=&B10100010 ' APAGAMOS PANTALLA
50 OPEN"GRP:"AS1
60 A$="M+5,-31E20F10M+6,+15D7M+10,+10M+6,+3
M+3,6L60"
70 DRAW"A0C10BM40,146S6"+A$
```

```
80 DRAW"AOBM130,145S8"+A$
90 PAINT(50,140),10
100 PAINT(145,144),10
110 CIRCLE(80,40),20,10
120 PAINT (80,40),10
130 CIRCLE(72,34),4,1:CIRCLE(88,34),4,1:PAI
NT(72,34),1:PAINT(88,34),1
140 CIRCLE(80,30),20,1,3.91,5.6
150 CIRCLE(82,100),20,12,,,4:PAINT(84,102),
12
160 CIRCLE(82,110),20,12,,,3:PAINT(84,120),
12
170 CIRCLE(72,110),20,12,,,4:PAINT(74,112),
12
180 CIRCLE(72,120),20,12,,,3:PAINT(74,130),
12
190 CIRCLE(180,20),20,14,,,.4:PAINT (182,22
),14
200 CIRCLE(170,20),20,14,,,.3:PAINT(154,22)
,14
210 FORI=200TO230STEP18
220 CIRCLE(I-10,130),16,12,,,4:PAINT(I-12,1
32),12
230 NEXTI
240 A$="C8M+30,-30ER88D30R1U2L118BM+20,+2RC
14NR99D40R99NU40":B$="BM-15,-15HC1U18L28D18
R28BM+4,+4FU24L36D24R36BM-58,+10GU30L15D30N
M-10,+20GR18NM+10,+20FU33L21D33"
250 LINE(0,146)-(255,179),12,BF:LINE(0,180)
-(255,191),1,BF:DRAW"C1S3BM92,132XA$;"
260 LINE(81,132)-(83,150),1,B
270 LINE(71,142)-(73,156),1,B
280 PAINT(110,118),8:PAINT(144,142),14:DRAW
"S3C1BM180,164XB$;"
290 A$="AOC7E16R70G16L70"
300 DRAW"BM6,170XA$;"
310 PAINT(40,168),7
320 B$="AOC8E16R70G16L70"
330 FORXY=1TO6
340 NEXTXY
350 COLOR15:PSET(66,156),12:PRINT#1," ▲▲
▲▲      ▲▲▲▲▲▲▲▲▲":PSET(66,164),12:PRINT#1
," ■■■■      ■■■■■■■■■":PSET(66,172),12:PR
INT#1," ■■■■       ■■■■■■■■■"
```



```
360 VDP(1)=&B11100010 ' ENCENDEMOS PANTALLA
.
370  SP="": FOR J=0 TO 31:READS
380 SP=SP+CHR$(VAL("&H"+S))
390 NEXTJ
400 SPRITE$(1)=SP
410 FOR XX=255 TO 0 STEP-1:PUT SPRITE 1,(XX
,170),14,1
420 NEXTXX
430 GOTO 410
440 CIRCLE(70,30),4,14:CIRCLE(90,30),4,14:P
AINT(70,30),14:PAINT(90,30),14
450 CIRCLE(70,30),4,1:CIRCLE(90,30),4,1:PAI
NT(70,30),1:PAINT(90,30),1
460 RETURN
470 DATAO,0,1,2,7,4,D,15,25,FD,E5,FC,F7,14,
8,0,0,C0,0,0,FF,1,D5,1D,15,55,D5,10,F7,14,8
,0
```

Hemos introducido la línea 40 bajo "Rem" para que vea primero como se crea normalmente un dibujo en basic. Para comprobar el efecto de apagado de pantalla, elimine el "Rem" y vuelva a ejecutar el programa.

Por último, veamos para qué sirve el bit siete del registro:

Introduzca en las líneas 40 y 360 las siguientes modificaciones:

Poner bajo "rem" la línea 40. Sustituir la 360 por Interval On

En la línea 10 borrar Interval On.

Añadir dos nuevas líneas:

```
425 Vdp(1)=&B01100010
455 Vdp(1)=&B11100010
```

Los efectos que contempla obedecen simplemente al estado encendido o apagado del bit siete. Cuando este bit está en estado lógico uno, el procesador de video adopta una configuración de 4K y cuando está a cero, de 16K.

Lo normal es que nunca se cambie pero nosotros hemos querido hacerlo para que vea que el Screen 2 también trabaja con caracteres de ocho por ocho puntos al igual que el modo gráfico I.

Cuando utilizamos los poderosos mandatos del basic MSX para gestionar gráficos, lo que hace el procesador es evitarnos la tediosa tarea de tener que definir dibu-

jos como si de un puzzle se tratara a base de modificar caracteres y colorear después cada una de las ocho rayitas que componen cada carácter.

El Registro 2 VDP(2)

Utiliza los cuatro primeros bits (0–3) para definir en bloques de 1024 octetos (1K), la dirección donde se situa el parámetro que hemos definido como zona de pantalla y que equivale a lo que en los manuales de referencia viene traducido como Tabla de nombres de patrones y que desde nuestro punto de vista da lugar a numerosas confusiones.

Los valores iniciales para cada modo de pantalla son los siguientes:

VDP(2)	Screen 0	Screen 1	Screen 2	Screen 3
V. Inicial	000	006	006	002

Se dará cuenta de que el máximo valor que podemos representar con cuatro bits será de 16 puesto que $2^4 = 16$. Pues bien, la dirección de comienzo a la que nos referimos vendrá dada por la siguiente operación:

```
PrintVDP(2)*1024
```

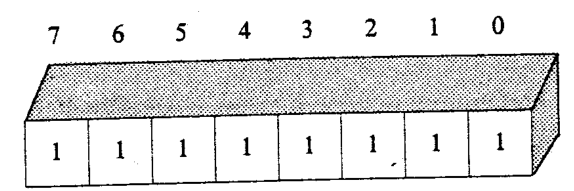
¿Recuerda de qué memoria dispone el procesador de video?

Tendremos pues, 16 valores posibles para el VDP(2), desde el 0 hasta el 15.

VDP(2)	DIRECCIÓN DE COMIENZO	
0	&H 0000	0
1	&H 0400	1024
2	&H 0800	2048
3	&H 0C00	3072
4	&H 1000	4096
5	&H 1400	5120
6	&H 1800	6144
7	&H 1C00	7168
8	&H 2000	8192
9	&H 2400	9216
10	&H 2800	10240
11	&H 2C00	11264
12	&H 3000	12288
13	&H 3400	13312
14	&H 3800	14336
15	&H 3C00	15360

Registro 3 VDP(3)

Define mediante los ocho bits que lo componen la dirección de la paleta de colores. Modificando el valor de este registro podemos situar la tabla de colores en la dirección de la VRAM que más nos guste. El valor máximo de este registro será de 255, (2^8).



Los valores iniciales para cada modo de pantalla son los siguientes:

VDP(3)	Screen 0	Screen 1	Screen 2	Screen 3
V. Inicial	000	128	255	000

La dirección de la paleta de colores podemos calcularla con la siguiente operación:

```
Print VDP(3)*64
```

Mención especial merece el caso del modo gráfico II. La razón la explicaremos en el apartado referido al modo de pantalla Screen 2.

Registro 4 VDP(4)

Este registro define mediante tres bits las direcciones posibles donde almacenar los datos que componen los 256 caracteres de que disponemos.

El valor máximo de este registro no puede exceder de 8, (2^3), ya que para definir los 256 caracteres empleamos 2048 octetos, (256*8) y la memoria disponible queda cubierta operando 2048 por 8.

La dirección de la tabla vendrá dada por la operación:

```
Print VDP(4)*2048
```

La representación de este registro con sus tres bits significativos puestos a uno quedaría de esta manera:

0	0	0	0	0	1	1	1

Los valores iniciales de que partimos para cada modo de pantalla son los siguientes:

VDP(4)	Screen 0	Screen 1	Screen 2	Screen 3
V. Inicial	001	000	003	000

Y las direcciones posibles de comienzo que podemos asignar se reducen a:

VDP(4)	DIRECCIÓN DE COMIENZO	
0	&H 0000	0
1	&H 0800	2048
2	&H 1000	4096
3	&H 1800	6144
4	&H 2000	8192
5	&H 2800	10240
6	&H 3000	12288
7	&H 3800	14336

Por ejemplo:

Si queremos desplazar la dirección base de la zona del generador de caracteres a la dirección 2048 solamente tendremos que hacer lo siguiente:

VDP(4)=1

Si queremos modificar los caracteres tendremos que dirigirnos a la nueva dirección dada por el valor del VDP(4).

Vamos, ahora, a realizar una prueba, combinando los tres registros que acabamos de ver.

```
10 SCREEN1
20 VDP(2)=1 ' PONEMOS EL COMIENZO DE LA ZON
A DE PANTALLA EN 1024
```



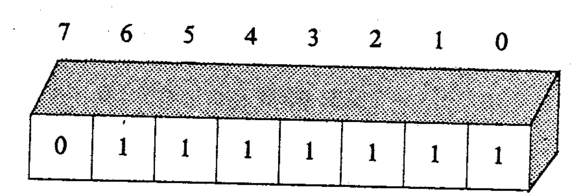
```
30 VDP(3)=28 ' PONEMOS LA TABLA DE COLORES
EN 28*64=1792
40 VDP(4)=1 ' PONEMOS LA TABLA DEL GENERADO
R DE CARACTERES EN 2048
45 FORI=0TO500:NEXTI
50 FORI=0TO767:VPOKE1024+I,65:NEXT
60 FORI=0TO1500:NEXT
70 SCREEN0
```

El ejemplo que acabamos de ver muestra como debemos utilizar combinadamente los registros 3, 4 y 5 del VDP para cambiar las tablas de la zona de pantalla, la paleta de colores y el generador de caracteres.

Notará que después de ejecutadas las líneas 20-40, la dirección de pantalla donde debemos escribir tendrá un valor distinto al usual. Observe la línea 50. En ella hacemos que se llene la pantalla con el código 65 correspondiente a la letra "A".

Registro 5 VDP(5)

Es el encargado de controlar mediante sus siete bits más significativos las direcciones donde comienza la tabla de atributos de figuras móviles.



Las direcciones de comienzo son las mismas para todos los modos de pantalla excepto el modo de texto que no los usa.

Los valores iniciales para cada modo de pantalla son los siguientes:

VDP(5)	Screen 0	Screen 1	Screen 2	Screen 3
V. Inicial	000	054	054	054

Con siete bits tendremos la posibilidad de escribir un valor máximo de 128 (2^7). Por tanto tendremos 128 valores posibles para este registro en bloques de 32*4=128 octetos.

Si queremos, pues, calcular la dirección de la tabla, escribiremos la siguiente operación: Print VDP(5)*128

Adelantemos ya que 32 son el número máximo de "sprites" que de hecho pueden coexistir a la vez en pantalla ya que sólo disponemos de 32 planos y que cada uno de ellos necesita cuatro octetos; dos de ellos para definir su situación en pantalla, otro para asignarle color y el cuarto como número identificativo de sprite.

Registro 6 VDP(6)

Define mediante tres bits la dirección donde comienza la tabla del generador de formas o patrones de las figuras móviles.

Los valores inciales para cada modo de pantalla son los siguientes:

VDP(6)	Screen 0	Screen 1	Screen 2	Screen 3
V. Inicial	000	007	007	007

Puede ser considerada como otra tabla de caracteres especiales y por tanto la extensión será análoga a la de la zona de caracteres, es decir, 256*8=2048 octetos que podemos situarlos en ocho posibles direcciones de la VRAM. (16384/2048=8). Damos la tabla a continuación.

VDP(6)	DIRECCIÓN DE COMIENZO	
0	&H 0000	0
1	&H 0800	2048
2	&H 1000	4096
3	&H 1800	6144
4	&H 2000	8192
5	&H 2800	10240
6	&H 3000	12288
7	&H 3800	14336

Dese usted cuenta que los llamamos especiales porque a pesar de poder tener definidos los 255, sólo podremos llevarlos a pantalla de 32 en 32. Esta limitación es compensada por su propiedad fundamental: moverse libremente en pantalla sin borrar los caracteres por donde pasa. De ahí que sean conocidos como "sprites" (espíritus).

```
10 SCREEN1
15 VDP(1)=&B11100001
20 VDP(5)=112 ' SITUAMOS LA ZONA DE ATRIBUT
OS DE SPRITES EN 14336
```



```
30 VDP(6)=5 ' SITUAMOS LA TABLA DEL GENERAD
OR DE SPRITES EN 10240
40 FORI=0TO7:VPOKE10240+I,VPEEK(2*8+I):NEXT
41 FORI=0TO7:VPOKE10240+32+I,VPEEK(198*8+I)
:NEXT
50 PUTSPRITE0,(100,100),11,0
60 VPOKE14340,116:VPOKE14341,106:VPOKE14342
,4:VPOKE14343,8
```

El ejemplo anterior es una muestra de como manejar las tablas de atributos y de generación de figuras móviles mediante los registros 5 y 6.

Le animamos a que practique sin miedo con los diversos registros dentro de los parámetros que hemos visto o incluso pruebe a introducir otros valores que a usted le parezcan.

Registro 7 VDP(7)

Los valores iniciales para los distintos modos de pantalla son los siguientes:

VDP(7)	Screen 0	Screen 1	Screen 2	Screen 3
V. Inicial	244	007	007	007

Este registro controla mediante ocho bits, los colores de tinta y fondo para el modo de texto (screen 0).

7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1

Los cuatro bits que vemos en el dibujo en estado lógico uno, controlan el color de fondo de todos los caracteres que aparecen en pantalla y los otros cuatro, los más significativos, hacen lo propio con el color de "tinta".

Para calcular las diversas combinaciones de colores entre fondo y tinta podemos utilizar esta fórmula:

color de tinta * 16 + color de fondo = valor a introducir en el VDP (7)

Ejemplo: Si queremos en modo de texto tener color de tinta negro sobre fondo azul claro, realizaremos lo siguiente

VDP(7)=16*1+7=23

Siendo 1 y 7 los códigos de negro y azul claro respectivamente.
Si realizamos la misma operación en otros modos de pantalla distintos al de texto obtendremos que el color de fondo será ahora color de borde.
Mediante la sentencia Basic Color tinta, fondo, borde usted ya accedía a este registro indirectamente.

```
10 SCREEN0
20 COLOR1,7
25 VDP(7)=&B00000000
30 LOCATE4,12:PRINT"HACIENDO PRUEBAS DE COL
OR"
40 FORI=0TO15:VDP(7)=I:FORJ=1TO90:NEXTJ,I
50 FORI=0TO15:VDP(7)=16*I:FORJ=1TO90:NEXTJ,
I
60 SCREEN1:GOTO 20 ' EN MODOS DISTINTOS AL
DE TEXTO SOLO CAMBIA COLOR DE BORDE
```

Registro 8 VDP(8)

Este registro es solo un registro de lectura que nos ofrece información sobre los "sprites" y las interrupciones.
Como registro de estado puede ser leido en cualquier momento; pero tenga en cuenta que su lectura pone a 0 el "flag" de interrupción que corresponde al bit 7 del registro.
El bit 6 es el flag que nos indica la presencia de más "sprites" de los autorizados en una misma horizontal. Se pone a uno cuando detecta la presencia del quinto "sprite". Se podría decir que es el guardian de la "Regla del quinto Sprite".
El bit 5 es el flag que detecta las colisiones entre "sprites". Es puesto a 1 cuando hay colisión.

LOS MODOS DE PANTALLA

1. Screen 0

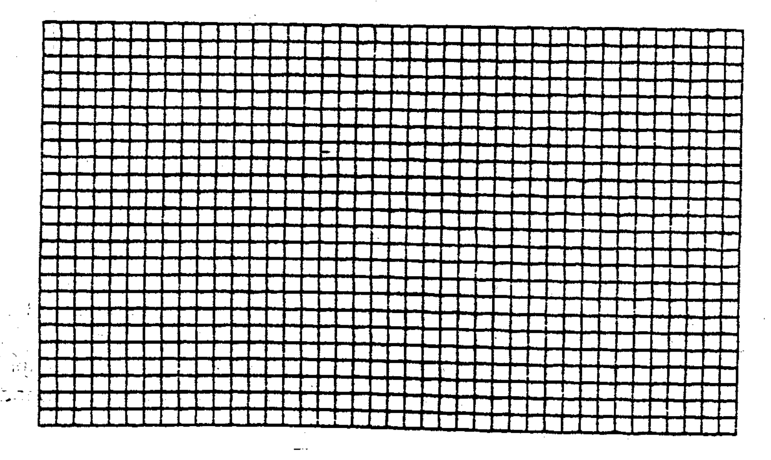
Este modo, el más sencillo, parco y limitado en todos sus parámetros, fue diseñado para modo de texto. Su especial característica es el uso de cuarenta columnas. No

obstante tiene otras características propias que son las que seguidamente vamos a analizar.
Como inicio, sería bueno echar una ojeada al estado inicial de los registros del VDP:

VDP.	BINARIO	DECIMAL	HEX
0	0 0 0 0 0 0 0 0	000	&H 0000
1	1 1 1 1 0 0 0 0	240	&H 00f0
2	0 0 0 0 0 0 0 0	000	&H 0000
3	0 0 0 0 0 0 0 0	000	&H 0000
4	0 0 0 0 0 0 0 1	001	&H 0001
5	0 0 0 0 0 0 0 0	000	&H 0000
6	0 0 0 0 0 0 0 0	000	&H 0000
7	1 1 1 1 0 0 0 0	244	&H 00f4

La división de la pantalla

Como sabemos, la pantalla para MSX tiene una resolución de 256 pixels horizontales por 192 verticales. Si dividimos ambos parámetros por 8 puntos tendremos la posibilidad de dividir la pantalla en 32 × 24 matrices de 8 × 8 puntos cada una. Pues bien, para el modo que en estos momentos analizamos y con objeto de dotar a la pantalla de 40 columnas; cada matriz ha sido estructurada en reducción a 6 × 8 puntos. Esto marca la especial característica del modo de texto.



Cuando enciende el ordenador y se pone a trabajar en BASIC, la anchura predefinida, por motivos estéticos es de 37 columnas por 23 filas; sin embargo con las instrucciones KEY OFF y WIDTH 40 podemos establecer las dimensiones antes señaladas.

La estructura de pantalla quedaría representada así, como una parrilla en la que cada matriz de 6 × 8 se corresponde biunívocamente con una posición de memoria en la zona reservada como archivo de pantalla.

Veamos ahora, las direcciones base de la tabla del procesador de video.

En nuestro ordenador las direcciones base de la tabla vienen dadas según el cuadro adjunto. Puede consultar en el suyo mediante la siguiente instrucción:

```
FOR I=0 TO 4:PRINT "BASE ";I;"=",BASE (I):NEXT
```

Observaremos que de las cinco tablas de que disponemos para cada modo de pantalla, el modo de texto solamente utiliza dos: la cero y la dos. La primera para la zona de pantalla y la segunda para el archivo de caracteres. En realidad, en el modo de texto no necesitamos más aditamentos.

BASE	DECIMAL	HEX	DIR COMIENZO
0	00000	&H 0000	ARCHIVO DE PANTALLA
1	--------	&H -----	TABLA DE COLORES
2	02048	&H 0800	GENERADOR CARACTERES
3	--------	&H -----	ATRIBUTOS DE SPRITES
4	--------	&H -----	GENERADOR DE SPRITES

El archivo de pantalla

Cada matriz de 6 × 8 puntos antes descrita ocupa un lugar determinado en la memoria de video a partir de la dirección &H00. Toda la zona reservada al archivo de pantalla ocupará una extensión de 40*32=960 casillas divididas en 24 grupos de 40 octetos cada uno.

Para comprobar la conexión directa entre lo que aparece en pantalla y el archivo de pantalla podemos teclear en mandato directo lo siguiente:

```
VPOKE 0,65
```

Veremos aparecer una "A" en el punto normalmente indicado por la sentencia BASIC: LOCATE 0,0. Esta casilla debe tener su posición correspondiente en el archivo de pantalla. En este caso es fácil saber que la casilla es la cero.

Basándonos en el hecho de que la pantalla está dividida en 40 columnas por 24 filas, es fácil deducir una fórmula que nos ayude a localizar la casilla correspondiente a un cuadratín proyectado en pantalla.


```
Base(0)+40*núm. de fila+núm. de columna
```

Ejemplo:

Si queremos proyectar un carácter en la posición representada por las coordenadas: sexta columna y cuarta fila, haremos la siguiente operación:

```
VPOKE Base(0)+40*3+5,núm. de código
```

Tenga usted en cuenta que empezamos a contar desde cero.

Tabla del generador de caracteres

Esta tabla comienza en nuestro ordenador en la dirección de memoria &H0800 y ocupa 256*8 = 2048 octetos (ocho octetos por cada carácter).

En estas celdillas consecutivas almacena el ordenador los datos necesarios para definir todos y cada uno de los caracteres de que dispone.

¿Cómo están organizados estos datos?

Cada carácter tiene asignada una zona de ocho octetos cuyos valores correlativos van dando forma a cada una de las ocho filas que posee el carácter.

		■						&B 00100000
	■		■					&B 01010000
■				■				&B 10001000
■	■	■	■	■				&B 11111000
■				■				&B 10001000
■				■				&B 10001000
								&B 00000000
								&B 00000000

Así, la "A" quedaría descompuesta en los ocho datos que figuran en la representación gráfica.

Para calcular la dirección de memoria en que están almacenados estos datos deberemos aplicar la siguiente fórmula:

```
Base(2)+cód.ASCII*8
```

En nuestro caso los octetos que definen la forma de la letra "A" estarán a partir de la posición &H800. Por tanto si deseamos conocer como está definida podemos hacerlo de esta manera:

```
FOR I=0 TO 7:?BIN$(VPEEK(2048+65*8+I)):NEXT
```

O si por comodidad los quiere sacar alineados:

```
10 FORI=0TO7
20 A$=BIN$(VPEEK(2048+65*8+I))
30 PRINTSTRING$(8-LEN(A$),"0")+A$
40 NEXT
50 END
```

Aplicaciones:

Combinando con un poco de imaginación lo que hasta ahora llevamos visto sobre el modo de texto puede usted encontrar numerosas aplicaciones. Aquí le exponemos algunas que pueden servirle de utilidad:

• Cómo crear caracteres nuevos

Para ver paso a paso la creación de un carácter, sitúese en modo directo y teclee los siguientes mandatos:

```
SCREEN 0
VPOKE420,1
```

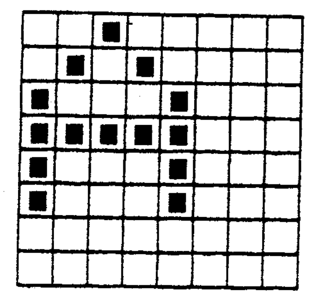
Usted verá aparecer en pantalla una cara sonriente, aunque no completa. Acuérdese de que el carácter está definido por una matriz de 8×8 puntos y que en screen 1 esa misma matriz queda reducida a 6×8 puntos siendo excluidas las dos columnas de la derecha.

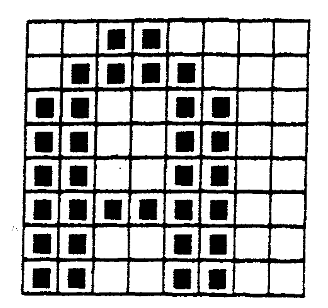
Nos vamos ahora a la dirección de comienzo de la tabla del generador de caracteres -Base(2)=2048- y puesto que el carácter que vamos a modificar es el núm. 1 en ASCII, teclearemos lo siguiente:

```
VPOKE 2048+1*8+0,&B00110000
VPOKE 2048+1*8+1,&B01111000
VPOKE 2048+1*8+2,&B11001100
VPOKE 2048+1*8+3,&B11001100
VPOKE 2048+1*8+4,&B11001100
VPOKE 2048+1*8+5,&B11111100
VPOKE 2048+1*8+6,&B11001100
VPOKE 2048+1*8+7,&B11001100
```

Si todo ha ido bien habrá visto cómo el carácter inicial se ha convertido en una "A" distinta de la que ya conocemos.

Vamos a comparar los dos caracteres:





Siguiendo este mismo procedimiento podemos hacernos con uno o más juegos de caracteres gráficos superpuestos a los ya predefinidos o coexistentes con ellos en otra zona de la misma Videoram.

• Rotación de caracteres.

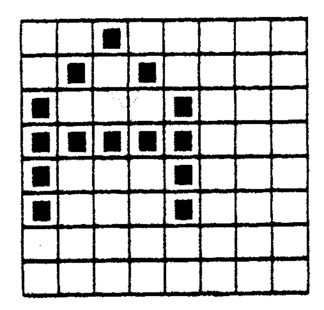
Para rotar un carácter deberemos rotar toda la matriz que lo compone. Para realizar esto en BASIC trataremos cada octeto de la matriz como una cadena y cogeremos sucesivamente un bit de cada cadena formando con ellos otra nueva cadena que se convertirá en el primer elemento de la matriz.

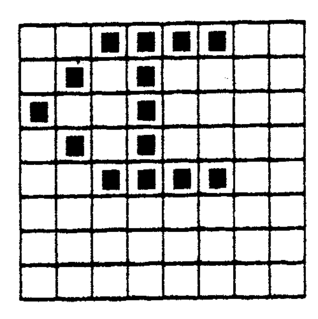
```
10 SCREEN0:CLEAR500:DEFINTA-Z
20 INPUT"NUMERO DE CARACTER";N
30 VPOKE30,N
40 FORI=0TO7:A$=BIN$(VPEEK(2048+N*8+I))
50 TV$(I)=STRING$(8-LEN(A$),"0")+A$
60 NEXT
70 FORA=1TO8:FORI=0TO7
80 X$=MID$(TV$(I),A,1)
90 Z$=Z$+X$
100 NEXTI
110 Y$(A)=Z$:Z$=""
120 NEXTA
130 FORI=1TO8:VPOKE(2048+N*8+(I-1)),VAL("&B
"+Y$(I)):NEXT
140 LOCATE10,4:PRINT"DATOS DEL CARACTER ROT
ADO"
150 FORI=1TO8:LOCATE15,5+I:PRINT"&H";HEX$(V
AL("&B"+Y$(I))):NEXT
160 LOCATE0,22:PRINT"¿QUIERES MAS?"
170 A$=INKEY$
180 IFA$="S"ORA$="s"THEN CLS:GOTO 20
```

```
190 IFA$="n"ORA$="N"THEN CLS:END
200 GOTO 170
```

Como verá, el proceso del programa primero carga una matriz con los datos del carácter introducido por el teclado y después realiza el giro en las líneas 70-120 para sacar el nuevo carácter en pantalla y por último ofrecer la composición en hexadecimal de los nuevos datos que lo componen.

Analizando este sencillo mecanismo, podrá hacer girar cualquier carácter en la dirección que necesite.





• Construir el "negativo" de un carácter.

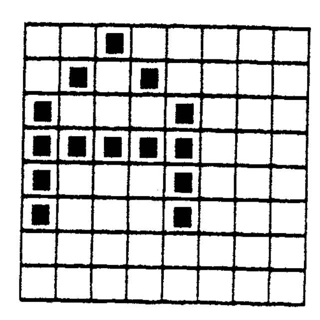
Otra de las aplicaciones que pueden serle útil, especialmente en el modo gráfico I podría ser intercambiar colores de tinta y fondo previa conversión de los bits encendidos en apagados y viceversa.

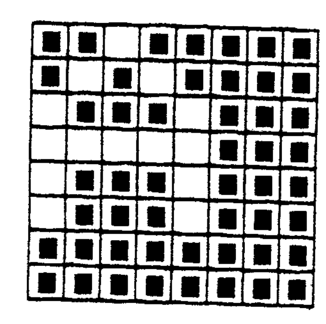
Observe en especial la línea 20 en la que con una simple resta se genera el "complementario" de cada dato.

```
10 FORI=0TO519:VPOKEI,1:NEXT
15 FORI=0TO7:A=VPEEK(2056+I):N(I)=A
20 A=255-A:F(I)=A ' CONVERSION DEL CARACTER
30 VPOKE2056+I,A ' SUSTITUCION POR LOS NUEV
OS VALORES
40 NEXT
50 FORI=0TO500:NEXT:BEEP ' RETARDO Y SEÑAL
SONORA
70 GOTO 15
```

Si quiere imprimir más rapidez al movimiento, añada una nueva línea al programa:

```
5 DEFINT A-Z
```



• Archivo de pantallas en otras zonas de la VRAM.

Otra de las más interesantes aplicaciones deriva del hecho de la infrautilización de la memoria de video en este modo de pantalla.

De los 16384 octetos de que disponemos, el modo de texto solamente utiliza 2048 para el generador de caracteres y 960 para el mapa de pantalla.

¿Qué sucedería si desplazamos el mapa de pantalla a otras zonas de la VRAM?

Si consultamos la tabla de las diferentes direcciones que podemos asignar al registro 2 (VDP(2)) y eliminamos los 2K que utiliza el generador de formas de los caracteres nos encontramos con 14 direcciones posibles y cada una de estas podría archivar una pantalla de texto entera.

```
1 DEFINT A-Z
5 FORI=0TO959:VPOKEI,1:NEXT ' LLENAMOS LA P
RIMERA PAGINA CON EL CODIGO ASCII 1
10 VDP(2)=1 ' CAMBIAMOS DE PAGINA
20 BASE(0)=1024
40 FORI=0TO959:VPOKE1024+I,65:NEXT ' LLENAM
OS LA SIGUIENTE PAGINA CON EL COD. ASCII 65
50 VDP(2)=4 ' CAMBIAMOS DE PAGINA
60 BASE(0)=4096
70 FORI=0TO959:VPOKE4096+I,67:NEXT ' LLENAM
OS LA PAGINA CON EL COD. ASCII 67
90 ' EXAMINAMOS LAS TRES PAGINAS
100 VDP(2)=1:FORI=0TO1500:NEXT:BEEP
110 VDP(2)=4:FORI=0TO1500:NEXT:BEEP
120 VDP(2)=0:FORI=0TO1500:NEXT:BEEP
130 IFINKEY$=""THEN100 ELSE VDP(2)=0
150 END
```

2. Screen 1

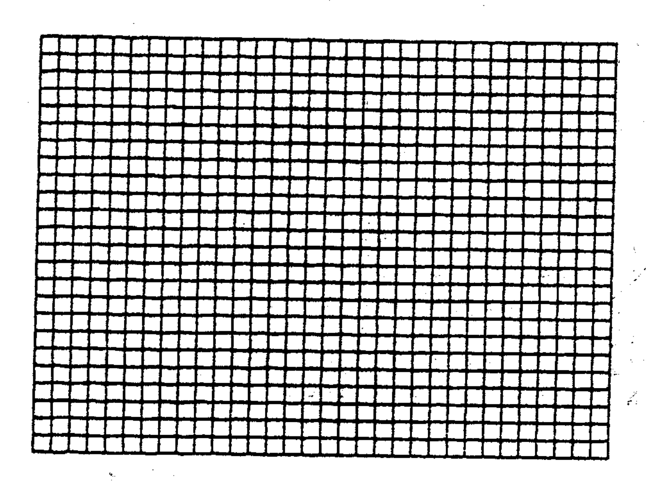
El modo de pantalla screen 1 es un modo de texto muy especial, y tan diferente al modo anterior, que merece ser llamado modo gráfico; si bien para diferenciarlo del modo screen 2 lo llamaremos reiteradamente Modo Gráfico I.

Veamos como primer paso el estado inicial de los registros del VDP.

VDP.	BINARIO	DECIMAL	HEX
0	0 0 0 0 0 0 0 0	000	&H 0000
1	1 1 1 0 0 0 0 0	224	&H 00E0
2	0 0 0 0 0 1 1 0	006	&H 0006
3	1 0 0 0 0 0 0 0	128	&H 0080
4	0 0 0 0 0 0 0 0	000	&H 0000
5	0 0 1 1 0 1 1 0	054	&H 0036
6	0 0 0 0 0 1 1 1	007	&H 0007
7	0 0 0 0 0 1 1 1	007	&H 0007

La división de la pantalla

En screen 1 tenemos dividida la pantalla en 32 × 24 matrices de 8 × 8 puntos cada una, es decir 24 filas de 32 caracteres.



A diferencia del modo de texto, tendremos una longitud de 768 octetos por pantalla en lugar de los 960 de que disponíamos antes.

La estructura de la pantalla quedaría representada por una "parrilla" en la que como es natural, cada matriz de 8×8 puntos se corresponde con una dirección de memoria en la zona reservada al archivo de pantalla.

Las direcciones base de la tabla del procesador de video vienen dadas en nuestro ordenador según el cuadro adjunto. Consulte en el suyo mediante la siguiente instrucción:

```
FOR I=0 TO 4:PRINT "BASE ";I+5;"=";BASE(5+I):NEXT
```

BASE	DECIMAL	HEX	DIR COMIENZO
5	06144	&H 1800	ARCHIVO DE PANTALLA
6	08192	&H 2000	TABLA DE COLORES
7	00000	&H 0000	GENERADOR CARACTERES
8	06912	&H 1b00	ATRIBUTOS DE SPRITES
9	14336	&H 3800	GENERADOR DE SPRITES

Observamos que tres de las cinco direcciones base de que dispone cada modo son activas y dos opcionales.

Por cuestión de método, veremos ahora las tres primeras y dejaremos las otras dos para el capítulo en que tratamos de las figuras móviles.

El archivo de pantalla

La diferencia con respecto al modo de texto que anteriormente hemos visto es, además de la división de la pantalla en 24 filas de 32 columnas, la dirección inicial de comienzo.

Aunque como ya sabemos la dirección base se puede alterar a nuestra propia voluntad, el archivo de la zona de pantalla viene inicializado en &H1800 66144 en decimal

Por tanto, conociendo estos preliminares, tanto si queremos escribir como leer un dato en una casilla determinada de las 768 de que disponemos, deberemos aplicar la siguiente fórmula:

BASE(5)+32 * NUM. DE FILA + NUM. DE COLUMNA

Tabla de colores

Una gran diferencia con respecto al modo de texto es la posibilidad de definir color de fondo y color de tinta para cada bloque de ocho caracteres.

La dirección de comienzo de la tabla base para el color viene dada por la variable BASE(6) que en nuestro ordenador comienza en &H2000 (8192 en decimal). La extensión de esta tabla es de 32 octetos (256 caracteres divididos en bloques de ocho).

¿Cómo están organizados estos datos?

El primer octeto de los 32 contiene el código de color de los ocho primeros caracteres (del 0 al 7) de la tabla ASCII que figura en el apéndice. El segundo contiene el código de los ocho siguientes, y así sucesivamente hasta completar los 32 octetos.

Para calcular el código de color que deseamos asignar a determinado bloque de ocho caracteres consecutivos deberemos aplicar el siguiente algoritmo:

VPOKE BASE(6)+(COD.ASCII DEL CARACTER\8),COD.COLOR

Siendo base(6) en nuestro ordenador &H2000 (8192 en decimal), le sumaremos la parte entera resultante de dividir el número de código del carácter que nos interese entre 8.

Una vez localizada la casilla que controla el bloque de ocho caracteres entre los que se encuentra el que queremos colorear, escribiremos en ella un número comprendido entre 0 y 255 que represente la combinación de colores de "tinta" y de fondo deseada.

La combinación de colores vendrá dada de operar por 16 el código de color deseado para la "tinta" del carácter, es decir, coloreará los bits "encendidos" del bloque correspondiente. A este número se le sumará el código de color que deseemos para el "fondo", esto es, los bits puestos a 0 en cada bloque.

Ejemplo:

Supongamos que queremos situar en la fila 10, columna 8 el carácter de código ASCII correspondiente al 215, coloreado con "tinta" azul claro y "fondo" gris.

Realizaremos esta tarea mediante tres pasos:

- Localizar la casilla deseada en la zona de pantalla.
- Escribir en ella el número 215.
- Localizar el bloque responsable del color de ese carácter y escribir en él un número que represente la combinación de colores buscada.

Los dos primeros pasos se resuelven con el mandato:

VPOKE6144+32*10+8,215

Y el tercer paso con:

Vpoke 8192+215\8,16*7+14

Pruebe ahora a darle diversos colores hasta encontrar el más adecuado a su gusto.

Por ejemplo: VPOKE 8192+215\8,16*1+15

Tabla del generador de caracteres

La dirección base de esta tabla viene dada por la variable BASE(7), que en nuestro ordenador comienza en la dirección &H00.

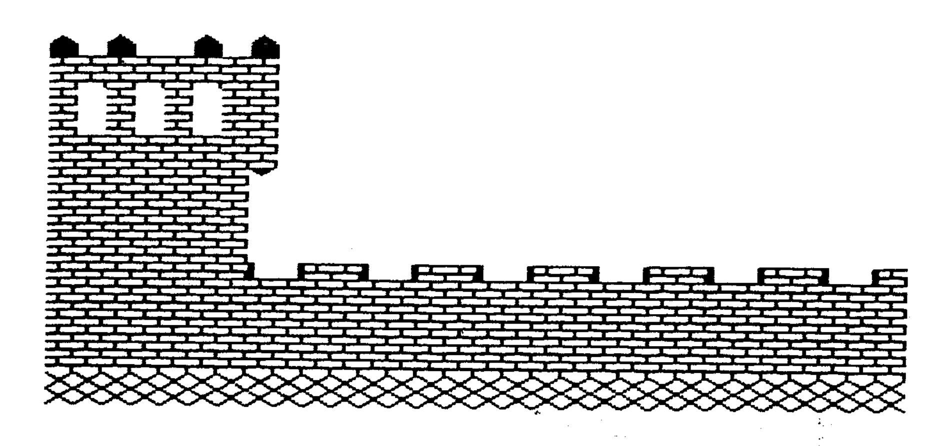
Como ya sabemos por el anterior modo de pantalla, ocupa un total de 2048 octetos. La limitación existente en el modo de texto queda aquí suprimida al visualizarse íntegramente cada carácter.

La diferencia parece poco significativa pero nos daremos cuenta inmediatamente de la ventaja que representa a la hora de realizar pantallas gráficas combinando adecuadamente las tablas del generador de caracteres y de colores.

Veamos un buen ejemplo de ello.

```
5 KEYOFF
10 SCREEN1:WIDTH31
20 COLOR 15,5,5
30 VPOKE8192+41\8,25
40 VPOKE8192+204\8, 16*6+5
50 VPOKE8192+194\8, 16*9+6
60 VPOKE8192+28\8, 16*1+12
70 A$(0)="":FORI=0TO29:A$(0)=A$(0)+CHR$(41)
:NEXT
80 A$(1)="":FORI=0TO6:A$(1)=A$(1)+CHR$(41):
NEXT
90 A$(2)=A$(1)+CHR$(41)
100 B$=CHR$(1)+CHR$(92):A$(3)="":FORI=0TO29
:A$(3)=A$(3)+B$:NEXT
110 FORI=0TO 7:READ A$
120 VPOKE 41*8+I,VAL("&H"+A$)
130 NEXT
140 FORI=0TO7:READA
150 A$(4)=A$(4)+CHR$(A)
160 NEXT
170 FORI=0TO7:READA
180 A$(5)=A$(5)+CHR$(A)
190 NEXT
200 FORI=0TO7:READA
210 A$(6)=A$(6)+CHR$(A)
220 NEXT
230 FORI=0TO22:READA
240 A$(7)=A$(7)+CHR$(A)
250 NEXT
260 PRINTA$(4):PRINTA$(5)
270 PRINTA$(2):PRINTA$(2)
280 FORI=0TO2:PRINTA$(6)
290 NEXT
```

```
300 PRINTA$(2):PRINTA$(2)
310 FORI=0TO5:PRINTA$(1) :NEXT
320 VPOKE6144+32*9+8,205
330 PRINTA$(1) +A$(7)
340 FORI=0TO5:PRINTA$(0):NEXT
350 PRINTA$(3):PRINTA$(3);
360 GOTO 360
370 DATA ff,10,10,10,ff,1,1,1
380 DATA 206,32,206,32,32,206,32,206
390 DATA 194,32,194,32,32,194,32,194
400 DATA 41,32,41,32,41,32,41,41
410 DATA 198,201,41,41,198,201,41,41,198,20
1,41,41,198,201,41,41,198,201,41,41,198,201
,41
```



También aquí, disponemos de "zonas muertas" en la memoria de video que pueden ser utilizadas como archivo para diversos fines: pantallas, caracteres, sprites, etc. y que pueden ser activadas en el momento necesario con la función VDP(X).

Podemos ver un ejemplo de como aprovechar esta facilidad introduciendo en el listado anterior las siguientes líneas:

```
360 FORI=0TO767:VPOKE7168+I,VPEEK(6144+I):N
EXT
361 CLS:FORI=1TO200:NEXT
365 BEEP:FORI=1TO10:VDP(2)=7:FORJ=1TO200:NE
XTJ:BEEP:CLS:VDP(2)=6:FORJ=1TO200:NEXTJ,I
```


En la línea 360 memorizamos el castillo a partir de la dirección &H1C00.

En la línea 365 hacemos un efecto de parpadeo que muestra la rapidez de acceso a la otra "página" archivada en &H1C00.

Es usted el que debe practicar y sacar el provecho oportuno a la hora de confeccionar sus programas.

Para finalizar esta visión rápida del modo gráfico I que será tratado posteriormente en profundidad, daremos aquí un juego de caracteres de texto (Numeración y letras Mayúsculas) que puede serle muy útil.

JUEGO DE CARACTERES

```
10 KEY OFF
20 SCREEN 1
30 VDP(4)=1 'CADA VEZ QUE UD QUIERA ACCEDER
 A ESTOS CARACTERES HAGA VDP(4)=1
40 FORI=0TO 79:READ A
50 VPOKE 256*8+48*8+I,A
60 NEXT
70 FORI=0TO 207:READ A
80 VPOKE 256*8+65*8+I,A
90 NEXT
100 DATA 28,38,99,99,99,50,28,0:' NUMERO 0
110 DATA 12,28,12,12,12,12,63,0
120 DATA 62,99,7,30,60,112,127,0
130 DATA 63,6,12,30,3,99,62,0
140 DATA 14,30,54,102,127,6,6,0
150 DATA 126,96,126,3,3,99,62,0
160 DATA 30,48,96,126,99,99,62,0
170 DATA 127,99,6,12,24,24,24,0
180 DATA 60,98,114,60,71,67,62,0
190 DATA 62,99,99,63,3,6,60,0
200 DATA 28,54,99,99,127,99,99,0 :' LETRA A
210 DATA 126,99,99,126,99,99,126,0
220 DATA 30,51,96,96,96,51,30,0
230 DATA 124,118,51,51,51,118,124,0
240 DATA 63,48,48,62,48,48,63,0
250 DATA 63,48,48,62,48,48,48,0
260 DATA 31,48,96,111,99,51,31,0
270 DATA 99,99,99,127,99,99,99,0
280 DATA 63,12,12,12,12,12,63,0
290 DATA 3,3,3,3,3,99,62,0
300 DATA 99,102,108,120,124,110,103,0
310 DATA 48,48,48,48,48,51,63,0
320 DATA 99,119,127,127,107,99,99,0
330 DATA 99,115,123,127,111,103,99,0
```

```
340 DATA 62,99,99,99,99,99,62,0
350 DATA 126,99,99,99,126,96,96,0
360 DATA 62,99,99,99,107,110,55,0
370 DATA 126,99,99,103,124,110,103,0
380 DATA 60,102,96,62,3,99,62,0
390 DATA 63,12,12,12,12,12,12,0
400 DATA 99,99,99,99,99,99,62,0
410 DATA 99,99,99,119,62,28,8,0
420 DATA 99,99,107,127,127,119,99,0
430 DATA 65,99,54,28,54,99,65,0
440 DATA 51,51,51,30,12,12,12,0
450 DATA 63,102,12,24,48,99,126,0
```

3. Screen 2

Este modo de pantalla es el ideal para cuando queremos insertar en nuestros cualquier realización gráfica que hayamos concebido.

Antes de pasar a estudiarlo con detalle, será bueno que repasemos los valores iniciales de los registros del VDP.

VDP.	BINARIO	DECIMAL	HEX
0	0 0 0 0 0 0 1 0	002	&H 0002
1	1 1 1 0 0 0 0 0	224	&H 00E0
2	0 0 0 0 0 1 1 0	006	&H 0006
3	1 1 1 1 1 1 1 1	255	&H 00FF
4	0 0 0 0 0 0 1 1	003	&H 0003
5	0 0 1 1 0 1 1 0	054	&H 0036
6	0 0 0 0 0 1 1 1	007	&H 0007
7	0 0 0 0 0 1 1 1	007	&H 0007

Las posibilidades que el BASIC MSX ofrece en este modo para realizar gráficos son "casi" profesionales. Cuando se utilizan los mandatos: line, circle, draw, pset, etc. con sus distintos parámetros, hacemos que el ordenador realice por nosotros un trabajo tan duro que de conocerlo en detalle no podría por menos de asombrarnos.

Seguramente usted ya habrá observado "cosas raras" cuando ha utilizado este modo a la hora de confeccionar alguna de sus pantallas. A veces, al trazar una línea superpuesta a un círculo ya pintado o sobre un Draw anterior etc., se observa una cierta distorsión que ocasiona una mezcla de colores poco armónica.

Estos defectos son difíciles de corregir. Algunas veces pueden ser subsanados cambiando los parámetros de las sentencias line o circle; otras cambiando el punto desde el que se traza una sentencia Draw, etc. Sin embargo otras veces lo más que permite nuestra paciencia es minimizar estos pequeños detalles para obtener desde BASIC una pantalla lo más aproximada a nuestra concepción original.

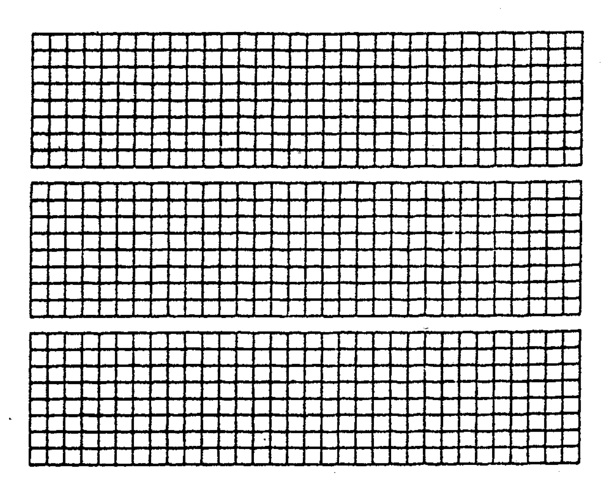
Quizá ya haya usted sospechado a qué se debe este fenómeno; incluso el manual de referencia de su ordenador puede que llame la atención acerca de la restricción de color existente para ocho puntos horizontales y la validez del último color especificado por una sentencia BASIC.

Lo que intentamos decir es que si en un mismo bloque horizontal de ocho puntos existe una línea u otro mandato similar que coloree esos ocho puntos de blanco, por ejemplo, y se intenta trazar por esa misma zona una línea de otro color, será este último el que predomine sobre el anterior; en este caso el blanco. A este fenómeno lo llamaremos "Preferencia del último mandato".

La división de la pantalla

El Screen 2 no es más que un Screen 1 muy especial, por eso aquí lo tratamos como modo gráfico II.

Al igual que el modo gráfico I, el área de pantalla está compuesta por una matriz de 32 por 24 cuadratines de 8×8 puntos, lo que nos da un total de 256×192 puntos numerados desde 0 hasta 255 y de 0 a 191 respectivamente.



La gran diferencia frente al modo gráfico I es que aquí la matriz de 32×24 cuadratines está dividida a su vez, por así decir, en otras tres "minipantallas" conocidas como ventanas de 32×8 cuadratines. Así, en una misma pantalla tendremos tres ventanas de ocho filas.

Es esta peculiar estructura de pantalla, además de otras características que vamos a ver, lo que hace posible la realización de gráficos complejos con lenguaje tan sencillo.

Las direcciones base de la tabla del procesador de video para este modo vienen dadas en nuestro ordenador según el cuadro adjunto. Compruebe usted con el suyo mediante el siguiente mandado:

FORI=0TO4:PRINT "BASE ";I+10;"=";BASE(10+I):NEXT

BASE	DECIMAL	HEX	DIR COMIENZO
10	06144	&H 1800	ARCHIVO DE PANTALLA
11	08192	&H 2000	TABLA DE COLORES
12	00000	&H 0000	GENERADOR CARACTERES
13	06912	&H 1B00	ATRIBUTOS DE SPRITES
14	14336	&H 3800	GENERADOR DE SPRITES

Como en el modo anterior, veremos ahora la composición de las tres primeras tablas.

El archivo de pantalla

Sobre una "parrilla" de pantalla exactamente igual que la del modo gráfico I, tenemos sin embargo una diferencia de concepto relativa a la división del bloque de 768 octetos en tres ventanas de 256 octetos cada una.

La operación para calcular la posición de cada octeto en la pantalla vendrá dada por una fórmula similar a la que utilizamos para el modo gráfico I, es decir, base(10)+(32 * número de fila) + número de columna. Para saber en todo momento en qué ventana de las tres disponibles se encuentra, dividiremos las 24 filas en tres bloques de ocho filas:

- Ventana 0 desde 0 hasta 7 filas
- Ventana 1 desde 8 hasta 15 filas
- Ventana 2 desde 16 hasta 23 filas

La fórmula rápida para hallar los comienzos de cada ventana será:

BASE(10) + (32*8) * Num. de ventana

Donde el número de ventanas estará comprendido entre 0 y 2. Esta división es muy lógica, como veremos en seguida, ya que la necesitaremos a la hora de combinar la tabla de caracteres y la de colores.

La tabla de colores

Como ya hemos visto por el apartado anterior, los 256 caracteres posibles del modo gráfico I, estaban divididos en 32 bloques de ocho caracteres cada uno a los que podíamos dar color de tinta y fondo por separado. En este modo, cada carácter puede ser coloreado independientemente y de manera diferente en cada una de las tres ventanas.

La limitación a dos colores para todo el carácter (tinta y fondo) sigue existiendo pero relativa a cada una de las ocho "rayitas" que lo componen. Esto es, ahora cada "rayita" de ocho puntos admite dos colores distintos. Como cada carácter está compuesto de ocho de estas "rayitas" -octetos-, podemos obtener hasta 16 colores distintos para cada uno.

Si tenemos presente que sólo puede haber dos colores por "rayita", comprendemos ahora que si se traza una línea de otro color que toque a otra anteriormente coloreada, algún color deberá desaparecer. Como en BASIC un nuevo mandato gráfico tiene preferencia sobre el anterior, será este último el que haga prevalecer su color.

Veamos un ejemplo harto significativo mediante el mandato BASIC: PSET.

```
10 COLOR1,14,14:CLS:SCREEN2
20 FORJ=0TO7:PSET(16*8,12*8+J),1:NEXTJ
30 FORI=1TO1000:NEXT
40 FORI=1TO10:BEEP:NEXT
50 FORJ=0TO7:PSET(16*8+3,12*8+J),12:NEXTJ
60 'SE PONE EN LA RAYITA AHORA UN NUEVO COL
OR
70 FORI=1TO1000:NEXT
80 FORI=1TO10:BEEP:NEXT
90 FORJ=0TO7:PSET(16*8+6,12*8+J),1:NEXTJ
100 FORI=1TO1000:NEXT
110 FORI=1TO10:BEEP:NEXT
120 CLS:GOTO 20
```

Observe como las sucesivas adiciones de puntos de distinto color en las ocho rayitas que componen el carácter ocasionan el solapamiento de los colores anteriores.

Para localizar el octeto inicial del bloque de ocho que dan color a un carácter procederemos de la siguiente manera:

Base(11)+núm. de código de carácter * 8 + Vn

Donde Vn será el resultado de la operación: (256*8)*n y n será el número identificativo de la ventana elegida (de 0 a 2).
Supongamos que se quiere hallar el comienzo del bloque de ocho que da color al carácter 65 en la ventana 2.
Preguntaremos el comienzo de la tabla de colores mediante el mandato: Print base(11). En nuestro ordenador la dirección es 8192. A este número le sumaremos 65*8 y como estamos en la ventana 2, le sumaremos (256*8)*2 obteniendo finalmente la dirección de comienzo del bloque de ocho octetos responsable del color del carácter 65.

Tabla del generador de caracteres

Disponemos de tres juegos distintos de 256 caracteres cada uno en correspondencia con cada una de las tres ventanas en que se divide la pantalla. Por tanto la extensión de la tabla del generador de caracteres será de 256*8*3 = 6144 octetos. Además, esta tabla está perfectamente sincronizada con la tabla de colores, de análoga extensión; de tal manera que cada octeto de la tabla del generador de caracteres tiene su exacta correspondencia en la tabla de colores.
Para localizar el octeto inicial de los ocho que componen un carácter se procederá de manera similar al modo gráfico I; teniendo en cuenta que un mismo código de carácter puede tener tres definiciones distintas dependiendo de la ventana en la que se encuentre.
La fórmula para localizar el comienzo de los ocho octetos responsables de la forma del carácter es análoga a la explicada para la tabla de colores. Usted sólo deberá sustituir Base(11) por Base(12) que es la dirección de comienzo del generador de caracteres.
Cuando se inicializa Screen 2, el contenido de la tabla del generador de caracteres está a cero ya que el ordenador los reserva para los mandatos gráficos ; por eso en el ejemplo siguiente lo primero que hacemos es cargar los caracteres desde la zona de archivo de la ROM donde reside permanentemente el juego de caracteres. La dirección de comienzo se halla en &H1BBF (7103 en decimal).

```
10 ' VENTANAS Y CARACTERES DE SCREEN 2
20 SCREEN2
30 FORI=0TO7:VPOKE65*8+I,PEEK(7103+65*8+I):
NEXT'DEFINIR EL CARACTER"A"EN LA PRIMERA VE
NTANA
40 FORI=0TO7:VPOKE256*8+65*8+I,PEEK(7103+66
*8+I):NEXT'DEFINIR EL CODIGO DEL CARACTER
"A" EN LA SEGUNDA COMO "B"
50 FORI=0TO7:VPOKE256*16+65*8+I,PEEK(7103+6
7*8+I):NEXT'DEFINIR EL CODIGO DEL CARACTER
"A" EN LA TERCERA COMO "C"
60 FORI=0TO7:VPOKE8192+65*8+I,31:NEXT 'DEFI
```



```
NIR LAS OCHO RAYITAS DEL CARACTER 65 COMO T
INTA NEGRA SOBRE FONDO BLANCO
70 FORI=0TO7:VPOKE8192+256*8+65*8+I,241:NEX
T 'DEFINIR LAS OCHO RAYITAS DEL CARACTER 65
 EN LA SEGUNDA VENTANA COMO TINTA BLANCA SO
BRE FONDO NEGRO
80 FORI=0TO7:VPOKE8192+256*16+65*8+I,244:NE
XT 'DEFINIR LAS OCHO RAYITAS DEL CARACTER 6
5 EN LA TERCERA VENTANA COMO TINTA BLANCA S
OBRE FONDO AZUL
90 FORI=0TO1000:NEXT  'retardo
100 FORI=0TO767:VPOKE6144+I,65:NEXT
150 GOTO 150
```

La gestión de pantallas en Screen 2

En el apartado anterior hemos visto cómo está distribuido el procesador de video al inicializar Screen 2. Cuando utilizamos este modo de pantalla para crear un gráfico, no solemos pensar en términos de "rayitas", ventanas, caracteres, etc., sino más bien en términos de mandatos gráficos del tipo: Line, Circle, etc. Sin embargo, el procesador sólo entiende de números contenidos en casillas de memoria. Por tanto, los mandatos BASIC deben ser traducidos de alguna manera a caracteres, posiciones de memoria, informaciones de color . . . .
Si a la hora de realizar una pantalla, la diseñamos en estos términos y "a mano", se puede tardar tranquilamente una semana en completarla frente a una tarde, como mucho, si utilizamos el BASIC. Cada sistema tiene sus ventajas e inconvenientes y debe ser usted el que decida cuál utilizar.
El diseño y perfección de las pantallas de un programa es determinante a la hora de su valoración. Por ello si utilizamos instrucciones cercanas al lenguaje directamente comprendido por el procesador, ganaremos calidad, rapidez y perfección en el diseño a cambio de emplear más tiempo en su realización.
Dada la cantidad de datos que tenemos que manejar en este modo y la facilidad y potencia de las instrucciones gráficas del BASIC MSX, preferimos estudiar el mecanismo de gestión de pantallas en CM en el modo gráfico I; y remitir al lector al apartado "El screen 1 en modo gráfico" del capítulo siguiente para realizar pantallas gráficas complejas.
El screen 2 está especialmente diseñado para la utilización de las rutinas de la ROM encargadas de traducir las sentencias BASIC que usted conoce.
Los siguientes ejemplos simulan las rutinas ROM de un mandato gráfico desde el BASIC.

```
10 CLS:COLOR1,1,1
20 SCREEN2
```

```
30 LINE(0,7)-(255,7),15
40 FORI=1TO500:NEXT:BEEP
50 FORI=0TO31:VPOKE8192+7+8*I,12*16:BEEP:NE
XT
60 FORI=0TO31:VPOKE8192+7+I*8,12*16:BEEP:NE
XT
70 GOTO 70
```

Se ha retardado en las líneas 50 y 60 la instrucción: Vpoke para que se aprecie cómo se van coloreando las rayitas de ocho puntos que componen la longitud de la línea.

Ahora vamos a realizar el mismo proceso en CM pero llenando de una en una las 6144 rayitas que componen la tabla de colores.

El siguiente programa carga una rutina en CM desde el BASIC aprovechando la rutina de la ROM &H004D.

```
10 SCREEN2
20 FORI=&HF500TO&HF50F
30 READ A$
40 POKEI,VAL("&H"+A$)
50 NEXT
55 DEFUSR=&HF500
56 A=USR(0)
60 GOTO 60
100 DATA 21,0,20,3E
110 DATA 8
120 DATA CD,4D,00,23,23,3E,38,BC,20,F4,C9
```

Hemos resaltado la línea 110 para que pueda cambiar allí el color (en hexadecimal).

A continuación le ofrecemos el listado en ensamblador de la rutina y una breve explicación de su funcionamiento.

```
F500            5           ORG    #F500
F500  210020   10           LD     HL,8192
F503  3E08     20  BUCLE:   LD     A,#8
F505  CD4D00   30           CALL   #004D
F508  23       40           INC    HL
F509  23       50           INC    HL
F50A  3E38     60           LD     A,#38
F50C  BC       70           CP     H
F50D  20F4     80           JR     NZ,BUCLE
F50F  C9       90           RET
```


En la línea 10 cargamos HL con la dirección de comienzo de la tabla de colores. En la línea 20 se carga el acumulador con 8 (número de color). A continuación acudimos a una llamada de la ROM que se encarga de escribir en la dirección de la VRAM dada por HL el dato contenido en el acumulador. En las líneas 40-50 se incrementa HL dos veces con 1 (HL=HL+2). Finalmente cargamos en el acumulador la parte alta de la dirección final de la tabla de colores (&H3800). La línea 70 es equivalente a una sentencia BASIC IF H=A THEN, es decir compara el valor de H con el de A. Como el registro H lleva la parte alta del número contenido en HL, hasta que el valor de H no coincida con el de A continuará el bucle dado por la línea 80.

Observe la sencillez de la programación en CM gracias a las facilidades de utilizar las rutinas de la ROM. En este caso la línea 30 equivale a un VPOKE HL,A.

Veamos ahora un ejemplo de cómo no debemos utilizar los mandatos gráficos. De él deduciremos la forma en que trabaja el Modo Gráfico II cuando traduce los mandatos BASIC.

```
10 ' LO QUE NO SE DEBE HACER
20 SCREEN2
30 CIRCLE(125,90),90,11
40 PAINT(125,90),11
50 FORI=5TO190 STEP 5
60 CIRCLE(5,5),I,15
70 CIRCLE(250,5),195-I,1
80 NEXT
90 FORI=1TO1500:NEXT
100 FORI=1TO10:BEEP:NEXT:CLS
110 ' LO QUE SE DEBE HCER
120 CIRCLE(125,90),90,11
130 FORI=0TO4:READA,B,C,D
140 LINE(A,B)-(C,D),11,BF
150 NEXT
160 PAINT(125,179),11
170 GOTO 170
180 DATA 57,35,192,142,77,16,171,34,78,143,
171,164,42,64,56,118,193,64,208,118
```

Aunque la primera parte de este ejemplo es un poco exagerada, muestra a las claras la forma en que la ROM trata los mandatos gráficos:

- Los mandatos del tipo Line, Draw y Circle trabajan en la zona de la tabla de caracteres modificándolos convenientemente.

• Los mandatos del tipo Paint, Line B y BF, Pset trabajan en la zona de la tabla de colores aprovechando la rutina del "Box-Fill", que rellena de color los espacios comprendidos entre los límites dados por caracteres de igual color de tinta y que comienza en &H1809 de la ROM.

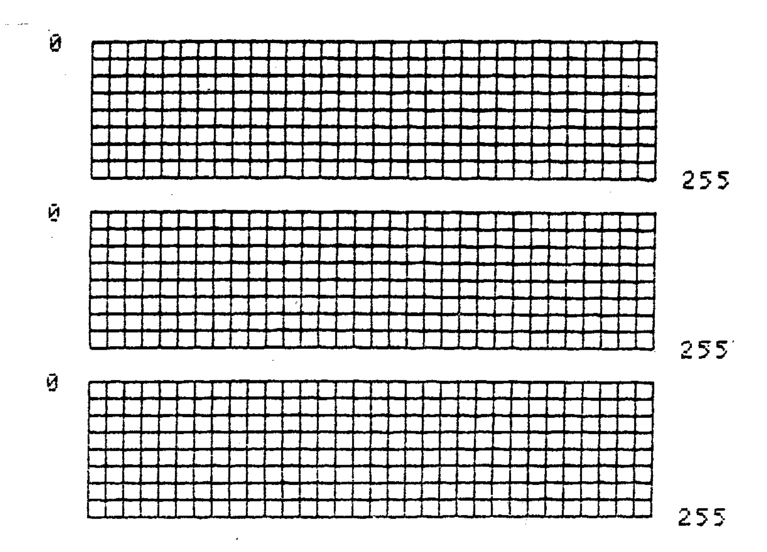
Nosotros le aconsejamos que cuando trate de dar color a grandes zonas de pantalla evite el Paint intentando aproximarlo con mandatos Line BF como se muetra en la segunda parte del ejemplo. Ya que Paint compara continuamente con la tabla de caracteres antes de colorear "rayita a rayita" la zona de la tabla de colores correspondiente. Por el contrario, Line BF rellena en bloque la zona comprendida entre los "caracteres inicial y final" dados por la primera y última coordenada de la tabla de colores.

A continuación veremos un ejemplo que explica cómo son tratados los textos en este modo y la razón por la cual debemos abrir un archivo previamente.

```
5 KEYOFF
10 OPEN"GRP:"AS1
20 COLOR1,1,1
30 SCREEN1:WIDTH32
40 COLOR11
50 A$=" \\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\////////
//////////////////////////////////////////////\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\/////////////////////////////////////////////////////
//////////////////////////////////"
60 LOCATE12,22:PRINT" SCREEN 1"
70 LOCATE0,0:PRINTA$
80 FORI=1TO1500:NEXT
90 FORI=1TO10:BEEP:NEXT
100 CLS:SCREEN2
110 PSET(12*8,22*8),1
120 PRINT#1," SCREEN 2";
130 PSET(0,0),4
140 COLOR 11:PRINT#1,A$
150 FORI=1TO1500:NEXT
160 FORI=1TO10:BEEP:NEXT
170 GOTO 30
```

Debido a la correspondencia total entre la tabla de caracteres y de pantalla; y en concreto entre cada ventana y su juego de caracteres correspondiente, la memoria de video no tiene espacio suficiente para tenerlos predefinidos y a la vez poder trabajar con instrucciones de tipo gráfico.

Por lo anteriormente dicho, debemos recordar al procesador, abriendo un archivo, que vaya a la zona de la ROM donde tiene archivado el juego de caracteres y redefina, por así decir, en el lugar de la pantalla que le corresponda, los caracteres dados a continuación de la instrucción PRINT #1, "—".



Esto es una desventaja con respecto al modo gráfico I, ya que aquí todos los elementos de una cadena son tratados independientemente.

Le recordamos el ejemplo del Castillo y la lentitud a la que se vería sujeto en modo gráfico II con Print #1,a$(x).

Para finalizar le vamos a dar un ejemplo de todo lo dicho ahora del modo Screen 2 mediante la realización de una pantalla gráfica desde el BASIC.

En este ejemplo encontraremos, además, tres rutinas en código máquina que nos serán muy útiles para el estudio de los capítulos siguientes.

Cuando ejecute el programa la pantalla quedará apagada en negro durante seis segundos. No se alarme. La explicación es que se utilizan dos llamadas a la ROM para evitar ver como se realiza la pantalla en BASIC. Esto ya lo aprendimos a hacer utilizando el bit 6 del VDP(1), ahora lo hacemos con llamadas respectivas a las direcciones &H41 para "apagar" y &H44 para "encender".

```
10 DEFUSR=&H41:DEFUSR1=&H44
20 COLOR1,13,1:SCREEN2
30 A=USR(0)
40 LINE(3,0)-(252,189),,B
50 LINE(3,10)-(60,50)
60 LINE(20,10)-(3,10)
70 LINE(20,10)-(70,50)
80 LINE(70,50)-(60,50)
90 LINE(20,0)-(20,10)
100 LINE(70,50)-(70,40)
110 LINE-(20,0)
```

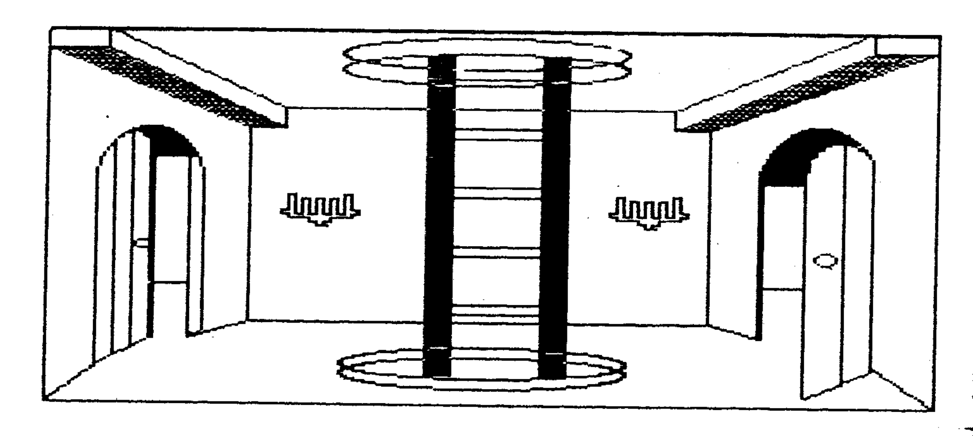
```
120 LINE(70,40)-(179,40)
130 LINE-(235,0)
140 LINE(1,191)-(60,149),4
150 LINE(60,148)-(60,50)
160 LINE(253,191)-(189,149),4
170 LINE(189,148)-(189,50)
180 LINE(235,0)-(235,10)
190 LINE-(252,10)
200 LINE(179,40)-(179,50)
210 LINE-(235,10)
220 LINE(179,50)-(189,50)
230 LINE-(252,10)
240 PAINT(10,13),1,1
250 LINE(50,0)-(77,20),1,BF:LINE(180,0)-(20
5,18),1,BF
260 PAINT(235,12),1,1
270 LINE(60,150)-(190,192),4,BF
280 CIRCLE(126,170),40,2,,,.2
290 PAINT(126,170),2,2
300 CIRCLE(126,175),40,1,,,.2
310 PAINT(126,169),1,1
320 CIRCLE(126,170),40,2,,,.2
330 LINE(78,0)-(180,39),,BF
340 CIRCLE(126,20),40,2,,,.2
350 PAINT(120,20),2,2
360 CIRCLE(126,12),40,1,.6,.4,.2
370 PAINT(126,14),1,1
380 CIRCLE(126,20),40,2,,,.2
390 LINE(110,13)-(117,176),,BF
400 LINE(142,14)-(149,176),,BF
410 LINE(142,14)-(149,176),2,B
420 LINE(110,13)-(117,176),2,B
430 FORI=20TO160STEP30
440 LINE(117,I)-(142,I+5),,BF
450 LINE(117,I)-(142,I+5),2,B
460 NEXTI
470 PAINT(45,1),1:PAINT(206,1),1
480 PAINT(50,191),4:PAINT(191,191),4
490  LINE(204,156)-(204,70)
500 LINE(235,172)-(235,65)
510 CIRCLE(220,82),34,1,.55,2.8,2
520 LINE(234,174)-(216,186),13:LINE-(216,16
5),13:LINE-(234,178),13:PAINT(220,172),13
530 CIRCLE(226,70),14,1,.8,3.2,1.5:CIRCLE(
222,115),3,1
```



```
540 LINE(227,56)-(227,179):LINE(203,166)-(2
15,146),4,BF:LINE(203,156)-(203,70):FORI=0T
02:LINE(207-I,145)-(207-I,157),13:NEXTI
550 LINE(204,77)-(215,77)
560 LINE(216,186)-(216,70):LINE(235,172)-(2
17,186)
570 PAINT(210,68),1,1
580 L$="S2;L2U4L4U4L12U4L4U4R4U16R2U1D1R2D2
0R4U16R2U1D1R2D16R4U16R2U1D1R2D16R4U16R2U1D
1R2D16R4U20R2U1D1R2D16R4D4L4D4L12D4L4D4R2"
590 DRAW"C10;BM172,100;"+L$
600 DRAW"C10;BM80,100;"+L$
610 LINE(17,173)-(17,75)
620 LINE(47,154)-(47,75):LINE(33,162)-(2,19
1),4
630 CIRCLE(32,80),30,1,.1,3.15,2
640 LINE(42,65)-(42,162),4
650 LINE(22,167)-(22,59):LINE(33,165)-(33,6
0),4:LINE(27,163)-(27,53)
660 CIRCLE(20,60),12,1,.06,.794,.9:CIRCLE(3
0,110),2.5,1
670 LINE(42,130)-(33,130),4
680 PAINT(37,150),4,4
690 LINE(44,65)-(33,65):LINE(17,172)-(32,15
8):LINE-(32,60):LINE(43,158)-(43,65):LINE(4
3,157)-(47,154)
700 LINE(5,188)-(32,163),4
710 PAINT(37,55),1,1
720 A=USR1(0):COLOR,,4
730 FORI=0TO767:POKE50000!+I,VPEEK(6144+I):
NEXT
740 FORI=1TO5000:NEXT
750 FORI=1TO1000:NEXT:FORI=1TO10:BEEP:NEXT
760 FORI=0TO767:VPOKE6144+I,190:NEXT
770 FORI=1TO1000:NEXT:FORI=1TO10:BEEP:NEXT
780 FORI=0TO767:VPOKE6144+I,PEEK(50000!+I):
NEXT
790 V=V+1:IFV<2THEN750
800 FORI=0TO37:READA$
810 POKE&HF500+I,VAL("&H"+A$)
820 NEXT
830 DEFUSR=&HF500:DEFUSR1=&HF50D:DEFUSR2=&H
F51A
840 A=USR(0)
850 FORI=0TO20:BEEP:NEXT:A=USR2(0)
```

```
860 FORI=0TO100:NEXT
870 FORI=0TO20:BEEP:NEXT:A=USR1(0)
880 FORI=0TO100:NEXT
890 GOTO 850
900 DATA11,C0,DA,21,0,18,1,0,3,CD,59,0,C9,2
1,C0,DA,11,0,18,1,0,3,CD,5C,0,C9,21,0,18,1,
0,3,3E,BE,CD,56,0,C9
```

Aprovechando el diseño de caracteres realizado por el programa, podemos transferirlo directamente a la impresora.



A continuación, adjuntamos el listado ensamblador de las tres rutinas en CM utilizadas en la última parte del programa y definidas en la línea 850.

```
F500                    ORG     #F500
                ;RUTINA DE ESCRITURA EN RAM
                ;DESDE VRAM
F500   11C0DA           LD      DE,56000
F503   210018           LD      HL,6144
F506   010003           LD      BC,768
F509   CD5900           CALL    #0059
F50C   C9               RET
                ;RUTINA DE ESCRITURA EN VRAM
                ;DESDE RAM
F50D   21C0DA           LD      HL,56000
F510   110018           LD      DE,6144
F513   010003           LD      BC,768
```



```
F516   CD5C00           CALL    #005C
F519   C9               RET
                ;RUTINA PARA ESCRIBIR UN DATO
                ;EN VRAM CIERTO NUM. DE VECES
F51A   210018           LD      HL,6144
F51D   010003           LD      BC,768
F520   3EBE             LD      A,190
F522   CD5600           CALL    #0056
F525   C9               RET
```

4. Screen 3

Este modo, conocido como multicolor, no es de muy frecuente utilización. Por eso lo vamos a explicar someramente según el esquema habitual.

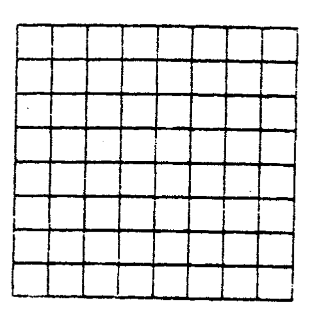
Los registros del VDP aparecen inicializados según los valores de la tabla adjunta:

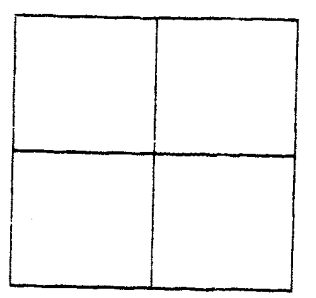
VDP.	BINARIO	DECIMAL	HEX
0	0 0 0 0 0 0 0 0	000	&H 0000
1	1 1 1 0 1 0 0 0	232	&H 00E8
2	0 0 0 0 0 0 1 0	002	&H 0002
3	0 0 0 0 0 0 0 0	000	&H 0000
4	0 0 0 0 0 0 0 0	000	&H 0000
5	0 0 1 1 0 1 1 0	054	&H 0036
6	0 0 0 0 0 1 1 1	007	&H 0007
7	0 0 0 0 0 1 1 1	007	&H 0007

La división de la pantalla

La pantalla está dividida al igual que en los dos modos anteriores, en 32 × 24 matrices de 2 × 2 puntos. Esta es la gran diferencia con respecto a los demás modos de pantalla y aquí radica la dificultad de este modo que parece tan complejo.

Cada punto de la matriz es equivalente a 4×4=16 puntos de los que estábamos acostumbrados a tratar.





A la vista del dibujo se observará fácilmente que los 256×192 puntos quedan reducidos a 64×48 puntos. Sin embargo, las coordenadas siguen siendo las mismas que en modo gráfico II por lo cual si consideramos el dibujo como el primer cuadratín de la pantalla, cualquier Pset comprendido entre (0,0) y (3,3) hará referencia a un mismo punto en screen 3.

Hecha esta salvedad, la parrilla de pantalla puede seguir siendo considerada a efectos de trabajo como una matriz de 32 filas por 24 columnas.

Las direcciones base de la tabla del procesador de video para este modo vienen dadas según el cuadro adjunto.

BASE	DECIMAL	HEX	DIR COMIENZO
15	02048	&H 0800	ARCHIVO DE PANTALLA
16	00000	&H 0000	TABLA DE COLORES
17	00000	&H 0000	GENERADOR CARACTERES
18	06912	&H 1B00	ATRIBUTOS DE SPRITES
19	14336	&H 3800	GENERADOR DE SPRITES

Tabla de colores

En realidad este modo se sirve de la tabla de caracteres para definir y dar color a los diversos patrones, por lo tanto, debemos considerar que no existe una tabla de colores propiamente dicha, tal y como hemos visto en los demás modos tratados hasta ahora.

Tabla del generador de caracteres

Al igual que en screen 1, la tabla viene dada por un juego de 256 caracteres de 8 octetos, siendo su longitud de 2048 bytes. Sin embargo la distribución para cada carácter es muy peculiar, ya que cada byte de los ocho que componen el carácter lleva una información de color similar a la que vimos en screen 1 para hallar los colores de "tinta" y "fondo".

Los dos primeros octetos que definen un carácter dan la información de los cuatro colores posibles de un cuadratín de dos por dos puntos de la tabla de pantalla para las filas 0, 4, 8, 12, 16 y 20.

Los dos siguientes octetos hacen lo mismo para las filas 1, 4, 9, 13, 17 y 21. Los dos siguientes a las filas 2, 6, 10, 14, 18 y 22; y los dos últimos a las filas 3, 7, 11, 15, 19 y 23.

La forma de hallar el primer octeto de los ocho que definen un carácter es análoga a la que ya conocemos.

Veamos un ejemplo aclaratorio.

```
10 SCREEN3
20 FORI=0TO767:VPOKE2048+I,32:NEXT 'Borrar
pantalla con 32
30 VPOKE65*8,16*5+1 'color del punto sup.iz
do.blanco y complementario negro
40 VPOKE65*8+1,16*1+15 'color del punto sup
.izdo. negro y complementario blanco
50 FORI=0TO767:VPOKE2048+I,65:NEXT
60 FORI=1TO500:NEXT:FORI=1TO10:BEEP:NEXT
70 VPOKE65*8,0:VPOKE65*8+1,0
80 VPOKE65*8+2,16*13+9
90 VPOKE65*8+3,16*9+13
100 FORI=1TO500:NEXT:FORI=1TO10:BEEP:NEXT
110 VPOKE65*8+2,0:VPOKE65*8+3,0
120 VPOKE65*8+4,16*1+11
130 VPOKE65*8+5,16*11+1
140 FORI=1TO500:NEXT:FORI=1TO10:BEEP:NEXT
150 VPOKE65*8+4,0:VPOKE65*8+5,0
160 VPOKE65*8+6,16*2+7
170 VPOKE65*8+7,16*7+2
180 FORI=1TO500:NEXT:FORI=1TO10:BEEP:NEXT
190 GOTO190
```

Esta peculiar distribución es debida al desajuste de 1 a 4 entre la tabla de pantalla y la del generador de caracteres; puesto que esta última define con ocho bytes la misma posición relativa a un carácter en pantalla que en screen 3 se define con dos.

Intuitivamente debe usted considerar que lo que antes era un carácter, ahora es 1/4 de carácter.

Utilidades del Screen 3

Quizá sea este modo el menos tenido en cuenta a la hora de programar, pero debemos decir en honor a la verdad que puede ser muy útil explotar algunas de sus características como por ejemplo la posibilidad de utilzar sprites, ocupar menos de 2K por cada página de pantalla, posibilidad de transferir pantallas a otras zonas de la VRAM, rotulación en letras gigantes, diagramas de barras con perspectiva tridimensional y otros.

Una de las características más resaltables de este modo es la posibilidad de colorear con el mandato Paint áreas de pantalla delimitadas con distinto color, algo imposible de realizar en screen 2 ya que para ello se necesitaba colorear el borde, pintar el interior del mismo color y por último dar otro color a los límites del área pintada.

Otra característica notable es la ausencia de los efectos producidos en screen 2 cuando se intentaba introducir un tercer color en una zona ocupada por una rayita con dos colores previamente definidos.

Capítulo 4

Las figuras móviles (Sprites)

1. Los Sprites

Uno de los aspectos que más resaltan, sobre todo en programas de juegos, es la impresión de movimiento lograda por ciertos gráficos especiales que aparecen como dotados de vida propia y que poseen, entre otras, las siguientes características:

- Son transparentes, es decir, pueden superponerse a cualquier gráfico previamente definido como si de un fantasma se tratara. De ahí el nombre de "Espíritu" (Sprite).
- No borran pantalla, esto es, a diferencia de un carácter normal que elimina al carácter sobre el que se superpone, estos otros lo solapan momentáneamente o incluso lo dejan entrever en las partes no definidas del Sprite.
- Pueden aparecer y desaparecer en cualquier zona de la pantalla e incluso intercambiarse entre sí o solaparse entre ellos mismos dando la impresión de que se ha generado otro sprite.

Pues bien, una de las facilidades más importantes que su MSX ofrece es precisamente la posibilidad de definir tantos Sprites como se necesitan dentro de los límites de memoria de la VRAM y todo ello con una pasmosa facilidad que sólo el BASIC MSX posee, a diferencia de otros ordenadores domésticos en los que hay que hacer complicadas manipulaciones para lograr efectos similares. Esta facilidad también tiene su correspondencia a la hora de programarlos en CM.

Prepárese a entrar en el maravilloso y "complejo" mundo de los sprites.

Qué es un Sprite

Seguramente ya tiene usted una idea formada de lo que es un Sprite. Nosotros a efectos de trabajo y para seguir el esquema lógico del funcionamiento del procesador de video, los trataremos como caracteres especiales debido a que la forma de construirlos es idéntica a la creación de caracteres vista en capítulos anteriores.

A primera vista, lo que acabamos de decir parece erróneo puesto que usted ha definido y trabajado con Sprites cuatro veces más grandes, esto es, de 16×16 pun-

tos. Pero si analiza con nosotros el proceso seguido, verá que realmente es como si hubiera definido cuatro caracteres por separado que, como si de un puzzle se tratara, formarán posteriormente el Sprite diseñado.

Desgraciadamente, ciertas limitaciones en el número de sprites que pueden coexistir simultáneamente en pantalla hace que en ocasiones haya que recurrir al efecto contrario, es decir, simular las características de un Sprite por medio de un diseño laborioso de caracteres que conlleva el tener que borrar previamente el área de la pantalla sobre la que se desplace el "Sprite simulado".

El procesador de video, como veremos en seguida, utiliza dos tablas para manipular estos "caracteres especiales":

- El generador de Sprites, donde están almacenados de ocho en ocho octetos los parámetros que definen la forma de cada Sprite.
- La tabla de atributos de Sprites que son 32*4 casillas que contienen la información necesaria para "portar" a pantalla la figura móvil elegida con sus coordenadas y su color.

A diferencia de los caracteres normales, estos otros, no están sometidos a la rigidez de las 32×24 posiciones posibles de un carácter en pantalla, sino que para ellos, debido a la impresión de movimiento con la que se les ha querido dotar, pueden situarse en cualquier posición de la pantalla.

2. Las tablas del VDP

En el capítulo tercero dejamos sin tratar, como recordará, las dos últimas tablas de cada modo de pantalla y referimos su estudio a este capítulo debido a que su valor inicial es común a todos los modos excepto el de texto que no los permite. Nos toca ahora abordarlas en conjunto.

Aunque como sabemos, podemos estructurar la VRAM a nuestra medida particular y por tanto reubicar las tablas relativas a los Sprites, su valor inicial para todos los modos es el siguiente:

BASES	DECIMAL	HEX	DIR COMIENZO
8,13,18	06912	&H 1B00	ATRIBUTOS DE SPRITES
9,14,19	14336	&H 3800	GENERADOR DE SPRITES

Tabla del generador de Sprites

Esta tabla contiene la información, en grupos de ocho octetos, del aspecto o tipo de 256 sprites diferentes al igual que vimos para la tabla de caracteres.

Como ya estará empezando a sospechar, el tratamiento que realiza el procesador es análogo al de los caracteres por lo cual para hallar la información de los ocho octetos que definen determinado número de Sprite procederemos de idéntica manera a como ya sabemos, es decir, multiplicaremos por ocho el número identificativo de Sprite, léase número de código de carácter en lugar de Sprite, y la equivalencia será total.

Aunque todo lo que llevamos dicho le parezca totalmente lógico, lo cual esperamos, se preguntará porqué complicarse tanto la existencia si parece más fácil asignar los 8 o 32 valores que dan forma a un Sprite, según sea el formato (8×8 o 16×16) a una variable alfanumérica o de cadena y posteriormente con la instrucción BASIC Sprite$ (n) = Cadena, hacer que el número de Sprite elegido sea igual a la cadena portadora del "diseño" elegido.

La razón de todo esto radica en que cualquiera que sea la forma elegida para llevar a cabo la tarea de creación de Sprites, el resultado final será siempre almacenar en la zona correspondiente de la tabla del generador de Sprites los valores asignados.

Hagamos un repaso de los posibles procedimientos BASIC para la creación de un Sprite:

```
10 'ESTE EJMPLO NO ES PARA TECLEAR SINO MER
AMENTE EXPLICATIVO.
20 '*****************************
30 SCREEN1,1:KEYOFF
40 'FORMA BASICA PASO A PASO
50 A$=CHR$(&HO)+CHR$(&HO)+CHR$(&H21)+CHR$(&
H42)+CHR$(&HFF)+CHR$(&H42)+CHR$(&H21)+CHR$(
&HO)
60 SPRITE$(0)=A$
70 PUT SPRITE 0,(100,20),15,0
80 FORI=1TO100:NEXT:FORI=1TO10:BEEP:NEXT
90 'FORMA BASIC MAS USUAL
100 SCREEN1,1
110 FORI=0TO7:READ A$
120 B$=B$+CHR$(VAL("&H"+A$))
130 NEXT
140 SPRITE$(1)=B$
150 PUT SPRITE 0,(100,20),12,1
160 GOTO 160
170 DATA 0,0,21,42,FF,42,21,0
```

Estas dos formas de construir Sprites se pueden considerar clásicas. La primera, con la que todos habremos empezado, es absurda a la hora de tener que construir numerosos Sprites. La segunda se presta más a ello ya que con un simple bucle se realiza el mismo proceso y además tiene la ventaja de poder definir, aumentando el bucle a 31, sprites de 16×16. Esta última forma es la que seguramente utiliza en

sus programas sustituyendo en la línea 140 Sprite$(1) por Sprite$(n) donde n será el número de sprites que se quieren definir. Por tanto insertando las siguientes líneas:

```
145 RETURN
500 FOR J=0 TO n
510 GOSUB 110:B$=""
520 NEXT
```

se obtiene una fórmula general muy práctica para la generación de N Sprites.

Nosotros, ahora, le pedimos que se "olvide" de todo eso. La razón estriba en que es más fácil, rápido y económico meter directamente en memoria los datos necesarios sin tener que pasar por la creación de variables de cadena u otros procedimientos similares.

La tarea anterior habría quedado simplificada de esta manera:

```
10 FOR I= 0 TO DATOS
20 READ A$
30 VPOKE 14336+I,VAL("&H"+A$)
40 NEXT
100 DATA ......
110 DATA ...
.... ETC.
```

Como la extensión de la tabla es análoga a la de caracteres, la variable DATOS podrá disponer de 2048 octetos −1, o un número mayor si queremos acceder por páginas a la VRAM. En definitiva, la variable DATOS será relativa a la cantidad total de datos a transferir.

El hecho de que el tamaño de los Sprites sea de 8×8 o de 16×16 no tiene excesiva importancia. Pero debemos tener en cuenta que en este último caso los datos estén dados en el orden correcto ya que el procesador entiende que debe tomarlos en bloques de 32 octetos. Lo cual hace que el número de Sprites se vea reducido a 256/4=64 formas posibles.

Tabla de atributos de Sprites

Definimos la tabla de atributos como una zona reservada de la memoria de la VRAM inicializada a partir de la posición &H1B00 con una extensión de 32×4=128 octetos y que especifica mediante cuatro octetos los datos necesarios para controlar cada una de las 32 figuras móviles de que podemos disponer simultáneamente en pantalla.

Los datos que comporta esta información se refieren a:

- Coordenada Vertical del "portor".
- Coordenada Horizontal del "portor".
- Número de Sprite a Portar.
- Color del Sprite a Portar.

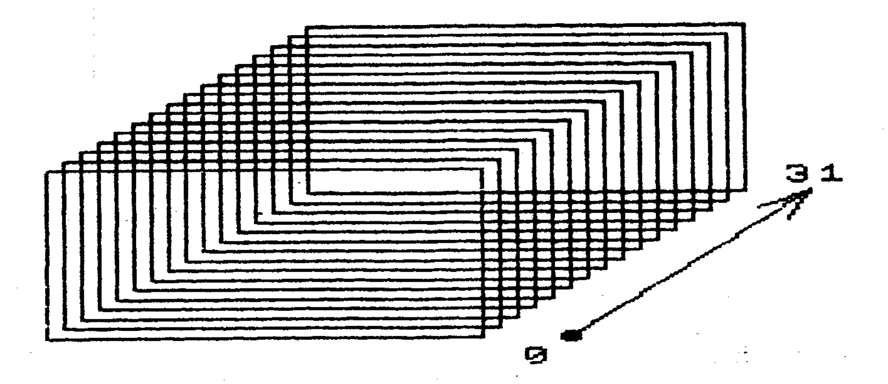
Hemos introducido un término que puede parecer un tanto raro pero que comprendido facilitará enormemente el concepto y uso de los llamados Sprites. ¿Qué entendemos por Portor?

Hasta ahora hemos seguido un mismo esquema para explicar el funcionamiento del Procesador de Video. Hemos comprendido como se diseñan y manejan los caracteres. Hemos hecho extensible el concepto aplicándoselo a los Sprites. Pues bien, a la hora de llevar a pantalla el número de Sprite deseado, nos encontramos con una limitación a la que no estábamos acostumbrados, ya que aunque podamos tener definidos hasta 256 formas distintas de Sprites o más, sólo podremos tener a la vez 32 de estas figuras.

Esta limitación es debida precisamente a que solo existen 32 Portores numerados de 0 a 31 encargados de "portar", de ahí el nombre, a la pantalla determinado número de Sprite.

Consideremos un Portor como una especie de bandeja invisible que lleva los cuatro parámetros necesarios de un Sprite. El contenido de esa bandeja puede ser cambiado y modificado a nuestra voluntad. Cada "bandeja" tiene asignado un número de preferencia que indica al procesador cual ha de visualizar en caso de que dos o más portores coincidan en una misma posición. Cuanto más bajo sea este número, de más preferencia gozará.

Considere ahora que cada bandeja se mueve en un plano distinto siendo el más cercano el plano más exterior o de número más bajo.



Sitúese en mandato directo y vea con nosotros todo lo tratado hasta ahora:

```
SCREEN 1:KEY OFF
```

Para no tener que crear sprites haremos coincidir la tabla del generador de caracteres con la del generador de Sprites.

```
VDP(6)=0
PUT SPRITE 0,(100,30),1,1
```

Observe como la instrucción Put Sprite ha situado la típica cara del juego de caracteres en las coordenadas 100,30. Pruebe a dar otros valores al último parámetro de la sentencia. Si el último "1" lo sustituímos por 65 veremos aparecer la letra A. Hasta aquí todo es análogo al tratamiento de caracteres.

La coordenada vertical del portor 0 que es el que estamos utilizando coincide con el primer octeto de la tabla de atributos de Sprites. Por tanto, un Vpoke 6912,n hará que el número de Sprite "portado" por el portor 0 se traslade hacia abajo o hacia arriba según el valor de n.

El segundo octeto controla la coordenada horizontal, por tanto un VPOKE 6913,n hará lo mismo que el anterior pero en sentido horizontal.

El tercer octeto es el responsable del número de Sprite elegido que en este caso puede ser cualquiera de los 256 caracteres ASCII. Pruebe con VPOKE 6914,66.

Y el cuarto octeto será el responsable del color del sprite regido por el portor 0, con el cual estamos trabajando.

A la vista de esto, podemos extrapolar y sacar la fórmula general que nos permita relacionar dentro de la tabla de atributos de Sprites donde comienza cada grupo de cuatro octetos para cada uno de los 32 portones de que disponemos.

B+(Np*4)

Donde B será el valor dado por la base 8, 13 o 18; o el valor que nosotros hayamos asignado manipulando el registro correspondiente del VDP.

Np será el número de portor deseado y comprendido entre 0 y 31.

El orden de colocación de los cuatro parámetros, será inverso al de la instrucción Put Sprite. Es decir, si damos por X la coordenada horizontal, Y para la vertical, C para el color y N para el número de Sprite; el orden de asignación será Y - X - N - C cuando tengamos que trabajar cargando directamente la tabla de atributos por medio de la instrucción Vpoke. Entendiéndolo de esta manera estará muy cerca del CM.

Siguiendo en modo directo, teclee:

```
VDP(1)=&B11100011
```

Como ya debe saber, hemos ampliado el tamaño de los Sprites y además los hemos seleccionado como de 16×16. Por tanto aunque los datos que tenemos no han

variado, el ordenador los agrupa de cuatro en cuatro siguiendo el orden de agrupación que usted mismo verá con la siguiente instrucción:

```
PUT SPRITE 31,(100,30),1,ASC("D")\4
```

Recuerde siempre este orden a la hora de cargar la tabla del generador de Sprites siempre que trabaje con Sprites de 16×16. Y tenga presente que con este tamaño el número de sprites se reduce a 64; por tanto el último parámetro de la instrucción Put Sprite no debe exceder de 63.

¿Sabría decirnos qué número contendrá la casilla de memoria correspondiente al portor 31 encargada de memorizar el número de Sprite a portar en pantalla, si la hemos cargado haciendo uso de la sentencia Put Sprite 31,(100,30),1,63, estando en modo de 16×16?

En este último caso, ¿por qué saldrá el mensaje "Illegal Function Call" si ponemos un número mayor de 63?

La explicación de todo esto reside en que con la instrucción Put Sprite se realiza la operación de colocar en las casillas correspondientes al portor elegido los parámetros contenidos en la sentencia BASIC, de una forma automática según el modo elegido. Si hemos elegido 16×16, la casilla que lleva la información del número de Sprite contendrá 63*4. Si intentamos poner 64, el intérprete intentará escribir de forma automática 64×4 lo cual sobrepasa la capacidad que un octeto puede almacenar.

Para que se adiestre en el uso directo, base imprescindible para el posterior trabajo en CM le proponemos el siguiente ejercicio:

Indicar la serie de operaciones necesarias, sin hacer uso de la instrucción Put Sprite, para situar en pantalla en las coordenadas 125,40, el Sprite dado a continuación a tamaño ampliado.

0 2
1 3

Utilice para ello el portor 25.

3. El diseño de Sprites

Como ya hemos demostrado, el diseño de un Sprite es exactamente igual que el diseño de un carácter. Por tanto no vamos a insistir en ello. Le remitimos al capítulo posterior y al programa gestor de pantallas para obtener los números que definan el tipo de la figura móvil.

En este apartado vamos a construir a modo de entrenamiento un Sprite de 16×16 y a manejarlo.

Pero antes conviene tener en cuenta algunas precisiones:

1. Normas para la visualización de figuras móviles.

- Cada portor de figura móvil sólo podrá visualizar un determinado número de Sprite.
- Cuando se solapen dos portores de figura móvil diferentes, el portor de plano exterior de menor número "tapará" al de plano más interno.
- Cuando coexistan horizontalmente más de cuatro patrones de figura móvil, el procesador se quedará con los cuatro de número más bajo o de planos más externos.
- Cuando se omita el código de color, el procesador "entenderá" como especificado el que tenga el primer plano.
- Cuando se omita el número de código de sprite se sobreentenderá el de número de portor especificado.

2. Los Sprites solamente admiten un color relativo a la parte definida de la figura móvil, es decir, los que hemos puesto a "1" en su diseño. Los puntos vacíos quedarán como transparentes. Podemos compaginar esto con el concepto de caracteres considerando que un Sprite sólo admite color de "tinta" pero no de fondo. Por tanto si queremos definir un Sprite multicolor deberemos diseñarlo superponiendo distintos números de Sprites con portores distintos, haciéndolos coincidir en la misma coordenada.

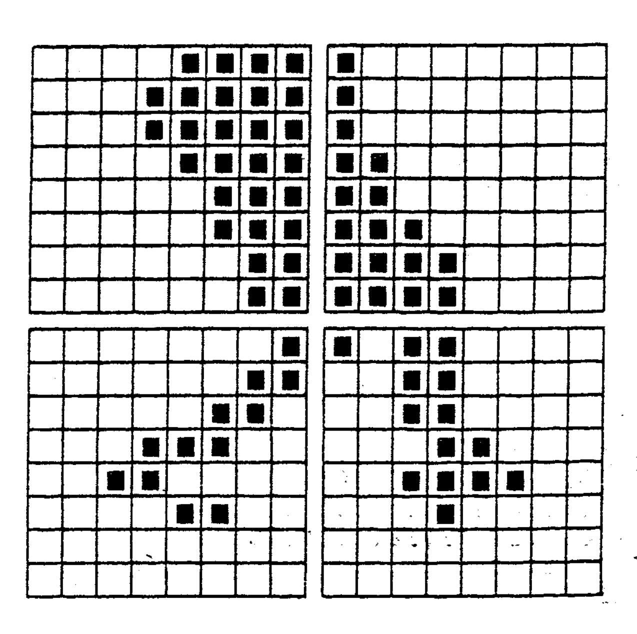
3. A la hora de construir Sprites de 16×16 deberemos hacerlo teniendo en cuenta que en realidad se componen de 4 Sprites de 8×8 donde el orden de agrupamiento viene dado según el dibujo adjunto:

1	3
2	4

Vamos a enfrentarnos con la tarea de realizar un Sprite multicolor complejo, quizá el más complejo que se pueda dar.

Este Sprite de 16×16 es la mitad del que vamos a realizar. Aunque aquí se represente como uno sólo, para lograr el efecto multicolor son necesarios los siguientes:

- Uno para el pelo. (Sprite 0)
- Otro para cara y manos. (Sprite 1)
- Otro para jersey y ojo. (Sprite 2)



En esta otra mitad, también han sido necesarios otros dos para dar efecto multicolor:

- Uno para la falda y los zapatos. (Sprite 3)
- Otro para las piernas. (Sprite 4)

En el diseño se debe tener en cuenta que las partes no definidas de un Sprite sean las que luego se llenen por medio de otro portor con el Sprite complementario, de tal manera que en la superposición final de los portores sean referidos a una misma coordenada.

```
5 SCREEN1,2
10 FORI= 0 TO 159
20 READ A$
30 VPOKE &H3800+I,VAL("&H"+A$)
40 NEXT
41 VPOKE 6912,19:VPOKE6913,99
45 X=4:FORI=0TO16 STEP4
50 VPOKE6917+I,100:VPOKE 6918+I,X
55 X=X+4:NEXT
60 VPOKE 6916,20:VPOKE6920,20
70 VPOKE 6924,35:VPOKE6928,35
80 FORI=0TO16 STEP4
82 READ A
85 VPOKE 6915+I,A
86 NEXT
90 FORI=1TO2000:NEXT
100 VDP(1)=&B11100011
110 BEEP:BEEP
115 FORI=0TO500:NEXT
120 VDP(1)=&B11100010
130 BEEP:BEEP
135 FORI=0TO500:NEXT
140 GOTO 100
490 ' PELO
500 DATA 3,7,3,5,d,1,7,F
510 DATA 1F,E,0,0,0,0,0,0
520 DATA 70,D8,80,0,0,80,0,80
530 DATA 0,0,0,0,0,0,0,0
540 ' CARA Y MANOS
550 DATA 0,0,0,0,0,0,0,0
560 DATA 0,0,0,10,30,60,0,0
570 DATA 0,0,50,FE,F8,70,0,60
580 DATA 0,0,3,6,C,8,0,0
590 ' JERSEY Y OJO
```



```
600 DATA 0,0,0,0,0,0,0,0
610 DATA 1,7,F,B,7,7,F,A
620 DATA 0,0,30,0,0,0,0,0
630 DATA 80,C0,E0,F0,D0,80,80,80
640 ' FALDA Y ZAPATOS
650 DATA 5,1F,1F,F,7,7,3,3
660 DATA 0,0,0,10,30,C,0,0
670 DATA 0,80,80,C0,C0,E0,F0,F0
680 DATA 0,0,0,0,2C,10,0,0
690 ' PIERNAS
700 DATA 0,0,0,0,0,0,0,0
710 DATA 1,3,6,C,0,0,0,0
720 DATA 0,0,0,0,0,0,0,0
730 DATA B0,30,30,18,10,0,0,0
740 ' COLORES
750 DATA 1,8,11,2,8
```

A continuación pasaremos a mostrar cómo podemos archivar en RAM la información de los Sprites que acabamos de realizar. Ya que los necesitaremos para explicar posteriormente el movimiento de los mismos.

Ya conoce la rutina de paso de datos de VRAM a RAM, también dispone de un modelo de cargador hexadecimal en BASIC. Pero en este caso, como ya tenemos los datos que componen los Sprites en VRAM, los transferiremos con un simple bucle a una dirección alta de la RAM normalmente no utilizada. Para lo cual volveremos al programa y cambiaremos la línea 30 por:

```
30 POKE 61.300+I, VAL("&H"+A$)
```

Ejecute el programa y tendrá los datos a partir de la dirección 61300. Para utilizar estos datos posteriormente, borre el programa y teclee el siguiente:

```
10 FORI=0TO12
20 RERAD A$
30 POKE 61280!+I,VAL("&H"+A$)
40 NEXT
100 DATA 21,74,EF,11,0,38,1,A0,0,CD,5C,0,C9
```

A continuación salve los datos tanto de la rutina como de los datos de Sprites de la manera siguiente:

```
BSAVE"CAS:MUJER",61280,61360,61280
```

Una vez hecho esto, cada vez que quiera acceder a estos datos, teclee lo siguiente:

```
SCREEN 1:BLOAD"CAS:MUJER",R
```

4. Movimiento de Sprites

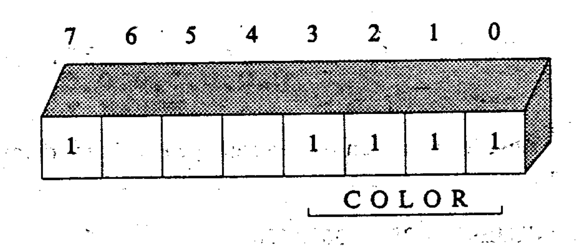
Para desplazar un Sprite lo único que tenemos que hacer es modificar el valor de las coordenadas vertical y horizontal del portor que "lleve" el número de Sprite deseado.

Teclee de nuevo VDP(6)=0 y practique dando valores, por ejemplo, a los octetos primero y segundo del grupo correspondiente al portor 0, que coinciden con los dos primeros de la tabla de atributos de Sprites.

Antes de dar movimiento a la figura de mujer que antes construimos conviene que tenga presente ciertas precisiones que quizá ya conozca:

- Las coordenadas vertical y horizontal de un Sprite toman siempre como referencia el punto superior izquierdo del mismo.
- La posición vertical ha sido previamente fijada para todos los portores en 209 con el fin de evitar su visualización en pantalla. Ésta es la forma más usual para hacer desaparecer un Sprite en esta coordenada. Si utilizamos la instrucción Put Sprite podrá escribirse un número negativo. El resultado lo interpretará sumando a 256 el número negativo que hayamos escrito.
- La posición horizontal ha sido previamente fijada para todos los portores en 255 ya que es el valor máximo permitido. Si damos un valor mayor el resultado será restarle a ese valor 256. La forma de hacer desaparecer los portores en la coordenada horizontal es poner en la instrucción Put Sprite el valor −32. Esto hará que la casilla de memoria correspondiente se ponga a cero y que el bit siete de la casilla responsable del color se ponga a 1, puesto que el octeto encargado de la coordenada horizontal no puede adoptar la configuración de octeto con signo.

En caso de que el contenido de la casilla responsable de la coordenada horizontal sea distinto de cero, el bit 7 de la casilla encargada del color originará un desplazamiento hacia la izquierda de 32 si está puesto a 1.



Para comprobar lo dicho, teclee:

```
VDP(6)=0
VDP(1)=&B11100011
VPOKE 6912,100
VPOKE 6913,100
VPOKE 6915,&B00001111
VPOKE 6915,&B10001111
```

Continuemos con la tarea iniciada anteriormente de dotar de movimiento a un Sprite multicolor gigante.

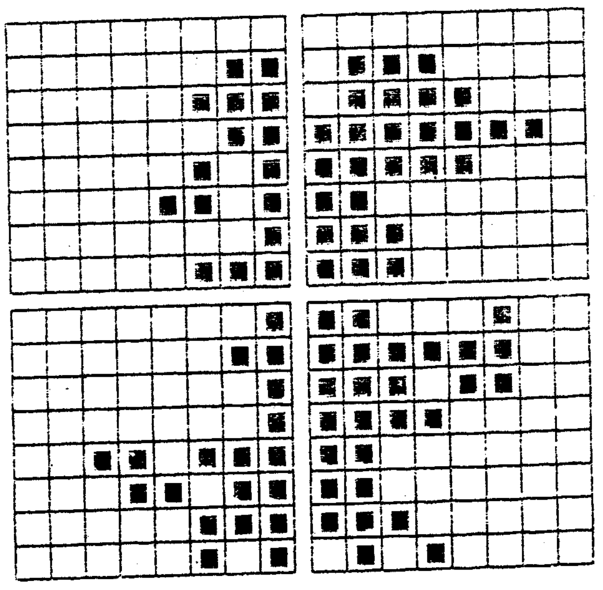
Como si de una película se tratara, la técnica consiste en sustituir los números de Sprite de los portores por otros números que representen el muñeco en otra posición. Por tanto definiremos ahora la siguiente posición de la "Señora". La impresión de movimiento se logrará sustituyendo una posición por la otra. Un movimiento será más perfecto cuanto más "fotogramas" compongan la secuencia de movimiento. Para el fin que nos hemos propuesto utilizaremos la mínima configuración; es decir, dos movimientos y repetición del proceso. Tenga en cuenta que muchos juegos comerciales que se precien contarán de tres, cuatro o incluso cinco posiciones distintas con lo cual la sensación de movimientos será mucho más real.

Para la posición dos de la "Señora" usaremos un proceso similar al anterior.

Los nuevos Sprites, por facilitar el movimiento, los colocaremos en posiciones contiguas de la VRAM a los que ya tenemos memorizados.

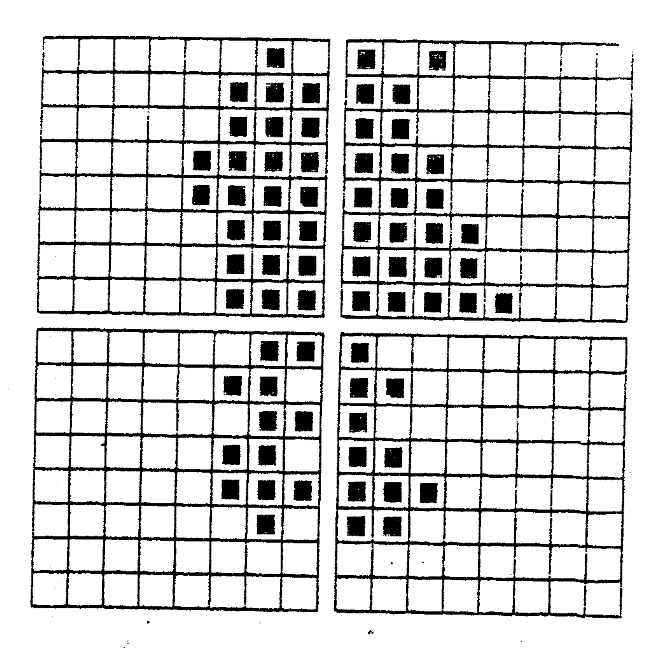
Recuperen los datos de la posición primera mediante:

```
SCREEN 1: BLOAD"CAS:MUJER",R
```



En este Sprite están superpuestos todos los planos portadores de Sprites de distinto color. Al igual que antes necesitamos:

- Un Sprite para cara y manos. (Sprite 5)
- Otro para jersey y ojos. (Sprite 6)
- El Sprite del pelo será constante en las dos posiciones. (Sprite 0)



Definimos, igualmente, las dos nuevas posiciones con dos nuevos Sprites de 16×16:

- Uno para falda y zapatos. (Sprite 7)
- Y otro para las piernas. (Sprite 8)

Antes de listar el programa conviene que compruebe si los datos relativos a la primera posición han sido transferidos correctamente a VRAM, para ello teclee:

PUT SPRITE 0,(100,100),1,0

Si vemos aparecer la cabellera negra de la señora, todo irá bien y podrá listar el siguiente programa que complete al anterior.


```
10 COLOR15,4,4:KEY OFF
20 VDP(1)=&B11100010
30 DEFINTA-Z:X=4
40 FORI=0TO127:READA$
50 VPOKE 14336+160+I,VAL("&H"+A$)
60 NEXT
70 FORI=0TO16 STEP4
80 READ A
90 VPOKE 6915+I,A
100 NEXT
110 VPOKE6912,19
120 VPOKE 6916,20:VPOKE6920,20
130 VPOKE 6924,35:VPOKE6928,35
140 FORI=0TO12 STEP4
150 VPOKE 6918+I,X
160 X=X+4
170 NEXT:BEEP
180 IF X=>40 THEN X=4
190 FORI=0TO16 STEP4
200 VPOKE 6913+I,Y
210 NEXT:BEEP
220 Y=Y+2:IFY>255THENY=0:Z=Z+8:GOSUB240
230 GOTO140
240 FORI=0TO16 STEP4
250 VPOKE 6912+I,VPEEK(6912+I)+Z
260 NEXT:BEEP
270 IFZ>47 THENZ=0:GOTO110
280 RETURN
290 ' CARA Y MANOS POSICION 2
300 DATA 0,0,0,0,0,0,0,0
310 DATA 0,0,0,0,30,18,0,0
320 DATA 0,0,68,FE,F8,C0,E0,E0
330 DATA 0,0,0,4,18,C,0,0
340 ' JERSEY Y OJO POS.2
350 DATA 0,0,0,0,0,0,0,0
360 DATA 1,3,1,1,7,3,7,5
370 DATA 0,0,10,0,0,0,0,0
380 DATA C0,E0,E0,F0,C0,C0,E0,50
390 ' FALDA Y ZAPATOS POS.2
400 DATA 2,7,7,F,F,7,7,7
410 DATA 0,0,0,0,5,3,0,0
420 DATA A0,C0,C0,E0,E0,F0,F0,F8
430 DATA 0,0,0,0,A0,C0,0,0
440 ' PIERNAS POSICION 2
```

```
450 DATA 0,0,0,0,0,0,0,0
460 DATA 3,6,3,6,2,0,0,0
470 DATA 0,0,0,0,0,0,0,0
480 DATA 80,C0,80,C0,80,0,0,0
490 ' colores
500 DATA 1,8,11,2,8
```

Como podrá comprobar, la impresión de movimiento, a pesar de estar dividida sólo en dos secuencias es bastante aceptable. Esto se puede mejorar moviendo los Sprites con una rutina en CM.

A continuación le daremos una rutina en CM específica para este caso. Comprobará que de la manera que hemos definido el movimiento en BASIC es muy fácil pasar a CM.

Antes que nada reunificaremos los dos movimientos en un solo bloque de datos de manera similar a como hicimos en el primero, colocándolos en posiciones contiguas.

Modifique la línea 50 como sigue:

```
50 POKE 61300+160+I,VAL("&H"+A$)
```

A continuación modifique el valor del registro BC con la suma total de los datos a leer (160+128=288), &H0120.

Para ello escribiremos la parte baja en 61287 y la parte alta en la casilla siguiente.

```
POKE 61287,&H20
POKE 61288,&H01
```

El siguiente paso será grabar en cinta los datos junto con su rutina de carga desde 61280 hasta 61590 con autoejecución en 61280.

El programa BASIC para mover las figuras quedará reducido eliminando las líneas que van desde la 40 a la 60 y desde la 290 a la 480.

A continuación insertaremos un programa cargador en CM que haga lo mismo que las líneas que van desde la 140 hasta la 230 excepto el desplazamiento vertical que comienza en la subrutina de las líneas 240 a 280.

```
101 ' CARGADOR BASIC DE LA RUTINA
    EN CM. MOVER "MUJER".
102 FOR I=&HEF10 TO&HEF52
103 READ A$
104 POKE I, VAL("&H"+A$)
105 NEXT
600 DATA E,0,3E,4,11,4,0,21
610 DATA 6,1B,CD,4D,0,C6,4,19
620 DATA FE,14,28,8,FE,24,28,2
```



```
630 DATA 18,F0,3E,4,F5,CD,47,EF
640 DATA 21,1,1B,6,0,79,CD,4D
650 DATA 0,19,4,3E,5,B8,20,F5
660 DATA C,CD,47,EF,F1,18,D0,D5
670 DATA 11,0,3,1B,7A,FE,0,20
680 DATA FA,D1,C9
```

Ejecute el programa resultante y observe el movimiento realizado en BASIC. A continuación interrumpa el programa y ejecute en mandato directo:

```
DEFUSR=&HEF10
A=USR(0)
```

¿Qué le parece la diferencia?

```
EF10                        ORG     61200
EF10  0E00                  LD      C,0
EF12  3E04                  LD      A,4
EF14  110400                LD      DE,4
EF17  21061B    REPITE:     LD      HL,#1B06
EF1A  CD4D00    VPOKE:      CALL    #004D
EF1D  C604                  ADD     A,4
EF1F  19                    ADD     HL,DE
EF20  FE14                  CP      20
EF22  2808                  JR      Z,GUARDAR
EF24  FE24                  CP      36
EF26  2802                  JR      Z,INICIO
EF28  18F0                  JR      VPOKE
EF2A  3E04      INICIO:     LD      A,4
EF2C  F5        GUARDAR:    PUSH    AF
EF2D  CD47EF                CALL    RETARDO
EF30  21011B                LD      HL,#1B01
EF33  0600                  LD      B,0
EF35  79        MOVER:      LD      A,C
EF36  CD4D00                CALL    #004D
EF39  19                    ADD     HL,DE
EF3A  04                    INC     B
EF3B  3E05                  LD      A,5
EF3D  B8                    CP      B
EF3E  20F5                  JR      NZ,MOVER
EF40  0C                    INC     C
EF41  CD47EF                CALL    RETARDO
EF44  F1                    POP     AF
EF45  18D0                  JR      REPITE
```

```
EF47   D5       290  RETARDO:   PUSH   DE
EF48   110003   300             LD     DE,768
EF4B   1B       310  BUCLE:     DEC    DE
EF4C   7A       320             LD     A,D
EF4D   FE00     330             CP     0
EF4F   20FA     340             JR     NZ,BUCLE
EF51   D1       350             POP    DE
EF52   C9       360             RET
```

Le pedimos disculpas si antes de leer este párrafo ha llamado a la rutina en CM ya que no se diseñó para retornar al BASIC. No obstante, en lugar de hacer uso del Reset, puede pulsar a la vez la siguiente combinación de teclas:

[CTRL]+[SHIFT]+[GRAPH]+[CODE] (LOCK)

Lo más lógico es que respire aliviado al ver aparecer el "amigable" Ok y volver a disponer del programa. La explicación de este "supermandato" es que se ha querido dotar de una función adicional de stop para salir de un bucle sin fin al comprobar una rutina en CM sin tener que hacer uso del "odioso" Reset. En efecto, si la casilla de memoria &HFBB0 contiene un número distinto de 0, podremos hacer uso de este mandato. En caso contrario no quedará más remedio que hacer uso del "Reset" o desconectar la alimentación del ordenador.

Dejando este pequeño inciso a un lado, que no deja de tener su utilidad, volvamos a la página anterior para comentar brevemente el listado ensamblador de la rutina de movimiento en CM.

Hemos intentado que la rutina fuera una "traducción" lo más exacta posible a su homóloga en BASIC para que vea la facilidad y la ventaja de trabajar en BASIC utilizando las funciones VPOKE y VPEEK a la hora de un paso posterior a CM.

En efecto, observe la similitud entre las línea 10-140 del listado ensamblador y sus homólogas 140-180 del listado BASIC. Igualmente entre las líneas 160-280 en ensamblador y las 190-230 del BASIC.

Observe como el registro DE realiza las funciones de la variable I de los bucles BASIC y cómo el acumulador (A) equivale a la variable X y realiza todas las funciones de comparación mediante la instrucción CP "simulando" las sentencias del tipo IF ... THEN del BASIC.

Finalmente, note en las líneas 150 y 260 como hemos tenido que ralentizar el movimiento con un bucle de retardo de 768 para evitar que el ojo humano sea incapaz de apreciar el movimiento. Le animamos a que cambie el valor de DE en la línea 300 según el retardo que desee o incluso suprima el retardo introduciendo en la casilla &HEF47 el valor &HC9 (RET) a ver qué pasa.

5. Puntos de la ROM para el tratamiento de Sprites

En este último apartado, le mostraremos las entradas de la ROM que faciliten el trabajo a la hora de mover los Sprites. Como ya hemos visto, podemos moverlos perfectamente pensando en términos de Vpoke y Vpeek, gracias a las entradas &H004D y &H004A ya vistas.

Añadimos ahora otras entradas de utilidad para poder manejarlos mediante los cursores o joysticks. Le animamos a dar sus primeros pasos en CM utilizando estas llamadas, ya que el movimiento en BASIC es muy lento y salteado si hay que mover muchos portores a la vez.

CALL &H00D5

Esta llamada es la que utiliza el intérprete para comprobar el movimiento del cursor o josyticks mediante la función STICK. Observará la gran similitud con el BASIC a la hora de utilizarla.

En efecto, esta rutina nos devolverá en el acumulador un número entre cero y ocho indicativo de la dirección del cursor o joystick que se está pulsando. Como ya sabe, estos números de salida se identifican con las ocho direcciones de movimiento posibles más el estado cero o de reposo:

0 No movimiento
1 Movimiento hacia arriba
2 Movimiento hacia arriba y derecha
3 Movimiento hacia derecha
4 Movimiento hacia abajo y derecha
5 Movimiento hacia abajo
6 Movimiento hacia abajo e izquierda
7 Movimiento hacia izquierda
8 Movimiento hacia arriba e izquierda

Ejemplo de ejecución:

```
STICK(0): LD A,0
          CALL &H00D5
          CP 1
          JR NZ,STICK(0)
          CALL MOVER
```

El anterior ejemplo muestra cómo podría ser la forma de averiguar si se está pulsando cursor arriba. Si es así, se llamaría a una rutina ficticia denominada MOVER, en caso contrario JR NZ devolvería la ejecución a STICK(0) para analizar de nuevo un posible movimiento.

Observe cómo esta rutina trabaja igual a la función BASIC: A=STICK(A).

Antes de la llamada deberemos cargar en el acumulador un 0, un 1 o un 2 según leer del cursor, del Joystick 1 o del Joystick 2.
A la salida, además de obtener en A el código del movimiento encontrado, variarán los contenidos de HL, BC y DE.

CALL &H00D8

Esta rutina también le será familiar, ya que la usa la función: STRIG(A) del BASIC.
En efecto, en este caso y previamente a la llamada, deberemos introducir en el acumulador un número indicativo del disparador a analizar, según el cuadro adjunto:

0 Barra espaciadora
1 Disparador 1 del Joystick 1
2 Disparador 2 del Joystick 1
3 Disparador 1 del Joystick 2
4 Disparador 2 del Joystick 2

A la salida obtendremos en el acumulador:

0 Si no se pulsa el disparador
255 Si se pulsa el disparador

Debido a la complejidad de esta rutina, también variarán los registros: BC, DE y HL.

CALL &H0069

Esta llamada inicializa todos los portores de Sprite, asignando color de Sprite a color de texto. La coordenada vertical toma el valor 209 para su no visionado en pantalla, y además se asocian sucesivamente número de portor con número de sprite (es decir, el portor 0 llevaría el número de sprite 0, etc.), alterando a la salida todos los registros (AF, BC, DE y HL).
Es una forma económica de borrar los Sprites de la pantalla, ya que durante el desarrollo de un juego es normal escribir mensajes diversos.

CALL &H0084

Esta llamada es utilizada por el BASIC en la instrucción SPRITE$. Se obtiene en el registro HL la dirección de VRAM a partir de la cual se encuentran memorizados los números que llevan la información de la figura del Sprite.

Ejemplo de ejecución:

```
LD A,0
CALL &H0084
```

Antes de llamar a la rutina, debemos introducir en el acumulador el número de Sprite seleccionado. En este caso obtendremos en HL 14336 que es la dirección de inicio de la tabla de Sprites. Acuérdese que podemos tener de 0 a 255 modelos de Sprites de 8×8 y de 0 a 63 de 16×16. El valor del acumulador deberá tener un número comprendido entre esos límites.
La llamada realizará el cálculo automáticamente, detectando si estamos en modo de 8×8 o de 16×16.
Debido al cálculo realizado, a la salida de esta rutina también nos variarán los registros AF, BC y DE.

CALL &H0087

Esta llamada es similar a la anterior. Pero ahora devolverá en HL la dirección de inicio de los cuatro atributos que definen a determinado portor de Sprite.
En el acumulador por tanto, deberemos introducir un número entre 0 y 31 relativo al número de portor deseado antes de acudir a la rutina.
Ejemplo de ejecución:

```
LD A,0
CALL &H0087
```

En este caso, obtendríamos en HL 6912 que es la dirección inicial de la Tabla de atributos de Sprite.
A la salida se modificarán además de HL, el registro DE y los flags.

CALL &H008A

Esta llamada nos devuelve el número de Bytes ocupados en la definición de un Sprite. Se podrán obtener por tanto, en el acumulador o un 8 (sprite 8×8) o un 32 (sprite 16×16).
La rutina sólo afectará al acumulador y a los flags, posicionándose el de acarreo a 1 si estamos en modo 16×16 o a cero en caso contrario.

Capítulo 5

La gestión de pantallas

1. Construcción de pantallas: BASIC y CM

El tema de la construcción de pantallas es bastante fácil tanto en BASIC como en código máquina. En este capítulo repasaremos las formas usuales que seguramente ya habrá utilizado en sus programas BASIC.

El siguiente ejemplo muestra la construcción de una pequeña pantalla en el modo gráfico I, mediante el manejo de cadenas. Esta forma es la más útil y rápida que poseemos utilizando al límite el BASIC MSX. En cuanto a la forma de definir las cadenas, hemos dividido el programa en dos partes:

La primera asigna diez cadenas en modo directo, mediante la introducción de caracteres gráficos en la misma.

La segunda forma es más lógica e interesante, ya que se archivan los números correspondientes de los caracteres utilizados en líneas data, que posteriormente serán leídos en su orden correspondiente.

La primera forma puede ser aconsejable si su ordenador posee en las teclas el diseño de los caracteres gráficos. La segunda forma es más cómoda y usual como es lógico suponer.

Observe como con el juego de caracteres estándar de la ROM se pueden realizar cadenas muy interesantes, dando un color adecuado a las mismas. Sin embargo, lo normal si se trata de hacer un programa de juegos es definir por el programador un juego nuevo de caracteres que facilite la construcción de las pantallas de acuerdo a la trama o guión del mismo.

```
10 KEYOFF:SCREEN1
20 ' construccion con cadenas
30 A$(1)="▲▼▲▼▲▼▲▼▲▼▲▼▲▼▲▼▲▼▲▼"
40 A$(2)="◘◘◘◘◆◆◆◆◆◆◘◘◘◘"
50 A$(3)="/\/\/\/\/\/\/\/\/\/\"
60 A$(4)="X*X*X*X*X*X*X*X*X*"
70 A$(5)="▄▀▄▀▄▀▄▀▄▀▄▀▄▀▄▀▄▀▄"
80 A$(6)="▀▄▀▄▀▄▀▄▀▄▀▄▀▄▀▄▀▄▀"
90 A$(7)=" ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ "
100 A$(8)="┬┼┬┼┬┼┬┼┬┼┬┼┬┼┬┼┬┼┬┼"
110 A$(9)="┴┼┴┼┴┼┴┼┴┼┴┼┴┼┴┼┴┼┴┼"
120 A$(0)="▪▮▪▮▪▮▪▮▪▮▪▮▪▮▪▮▪▮▪▮"
```

```
130 GOTO 210
140  ' CADENAS DEF. MEDIANTE DATAS
150 CLEAR500:CLS
160  RESTORE:FORI=0TO9:READJ,J1:IFJ<32TH
EN180
170  A$=CHR$(J)+CHR$(J1):GOTO190
180  A$=CHR$(1)+CHR$(JOR64)+CHR$(1)+CHR$
(J1OR64)
190  FORK=0TO9:A$(I)=A$(I)+A$:NEXT
200  NEXTI
210 ' RUTINA DE IMPRESION
220 VPOKE8192+207\8,&H17:VPOKE8192+199\8
,&HAC:VPOKE8195,&H6F
230 VPOKE8194,&HD9:VPOKE8192+208\8,&HF7:
VPOKE8193,&H7F:VPOKE8223,&HE2
240 FORI=0TO9:R=INT(RND(-TIME)*10)
250 LOCATE5,12+I:PRINTA$(R):NEXT:FORI=0T
040:BEEP:NEXT
260 FORI=0TO9:LOCATE5,23:PRINTA$(I):NEXT
270 DATA 196,254,205,206,209,210,29,30,2
8,15,199,193,193,199,208,207,21,18,21,17
```

Si le cuesta encontrar los caracteres gráficos, teclee sólo a partir de la línea 140. El resultado va a ser el mismo ya que el algoritmo es el adecuado; en este caso debe tener especial cuidado en transcribir exactamente la línea de data 270. Ejecutando varias veces el programa, verá que la impresión de las cadenas es elegida aleatoriamente por el ordenador en la primera fase.

El ejemplo es suficientemente aclaratorio a la hora de realizar pantallas; aunque salta a la vista que si tuviéramos que memorizar 20 pantallas completas en líneas data, utilizando el BASIC, no quedaría memoria para nada más. La solución no es otra que recurrir al código máquina.

Antes de proponerle algunos sistemas y ejemplos de memorización, conviene hacer un alto y llamar su atención sobre las técnicas empleadas en la construcción de programas comerciales.

Siendo éste un libro que pretende introducir al lector en la barrera que separa el BASIC del CM deberá explicar técnicas sencillas, fácilmente comprensibles y de uso general: excluyendo, por tanto, técnicas como la mal llamada "Filmation" o algoritmos extraños para comprimir pantallas. Nos proponemos pues, incitarle a la investigación intentando lo más sencillo y llevándole así, a la comprensión de lo más complicado.

Comparemos para empezar dos juegos típicos que seguramente ya conoce: el "Manic Miner" y "H.E.R.O". En el primero existen 20 pantallas completamente

distintas unas de otras, en cambio, en el segundo aparecen un número enorme de pantallas que parece no tener fin.

En el primer caso comprendemos facilmente que cada pantalla está memoriazada completa en RAM; de ahí su número limitado.

En el segundo caso, si no dejamos que nuestros ojos nos engañen, nos daremos cuenta de que unas cavernas son muy parecidas a otras, y además la secuencia de colores se repite cada cuatro niveles. En este juego se utiliza un algoritmo que solamente guarda en memoria los caminos a seguir por el personaje, con sus paredes y objetos. Una vez identificado el túnel a seguir, se llena la pantalla sobrante con una secuencia de números parecida al algoritmo de cadenas en data del ejemplo anterior. Interesante y hábil ¿verdad?

Las diversas técnicas surgen siempre en la búsqueda de la solución más lógica para un problema determinado. En la curiosidad que siente se encuentra el trampolín que quizá le lleve a realizar una técnica nueva y a darle nombre.

Le proponemos ahora el método básico de archivo de pantallas como base esencial para iniciarse en ese camino.

```
10 'Forma BASIC Manic Miner
20 KEY OFF:SCREEN1
30 DE=6144 ' DESTINO VRAM
40 HL=&H4000 ' DIR.INICIAL DE ARCHIVO DE PA
NTALLA EN RAM
50 BC=32*20 ' CONTADOR DE LONGITUD
60 VPOKE DE,PEEK(HL)
70 DE=DE+1
80 HL=HL+1
90 BC=BC-1
100 IF BC<>0 THEN 60
110 FOR I=0TO40:BEEP:BEEP:NEXT
120 ' RUTINA CM.
130 CLS:FORI=0TO40:BEEP:BEEP:NEXT
140 FORI=0TO12:READ A$
150 POKE&HF500+I,VAL("&H"+A$)
160 NEXT
170 DEFUSR=&HF500: A=USR(0)
180 FORI=0TO3000:NEXT:RUN
190 DATA 11,0,18 ' DE=6144
200 DATA 21,0,40 ' HL=&H4000
210 DATA 1,80,02 ' BC=32*20 (=&H0280)
220 DATA CD,5C,0 ' CALL BIOS adecuado
230 DATA c9 ' RET a BASIC
```

El ejemplo es una muestra de la sencillez de esta técnica y a la vez aclaración de las diferencias entre BASIC y CM en cuanto a rapidez y gasto de memoria. No

se preocupe si no ha visto ninguna pantalla maravillosa, ya que se ha utilizado la ROM del ordenador como un archivo ficticio de pantalla en la dirección &H4000.

En el siguiente apartado le enseñamos los CALL BIOS necesarios para completar la técnica del manejo y archivo de pantallas.

2. Entradas de la ROM que facilitan la tarea

CALL &H004A

Utilice esta llamada cuando quiera obtener en el acumulador el valor de determinada casilla de la Videoram apuntada por HL.

Su analogía en BASIC es similar a:

A=Vpeek (HL)

Ejemplo de ejecución:

```
LD HL,6144
CALL &H004A
```

El acumulador tendrá el Vpeek(6144). Esta llamada sólo afecta a los flags y al acumulador.

CALL &H004D

Esta llamada ocasiona el fenómeno contrario a la anterior. Su equivalencia en BASIC como es obvio, es:

Vpoke HL,A

Ejemplo de ejecución:

```
LD HL,6144
LD A,32
CALL &H004D
```

La casilla 6144 de la Videoram contendrá el valor 32.

CALL &H0056

Escriba a partir de la dirección de VRAM dada por HL, un cierto dato que deberemos introducir en el acumulador, tantas veces como indique BC.

Ejemplo de ejecución:

```
LD HL,6144
LD A,32
LD BC,768
CALL &H0056
```

Equivale a un CLS, ya que escribe el carácter 32 (código ASCII del espacio) 768 veces, empezando en la casilla de VRAM 6144 y acabando en la 6144+767.

A la salida, esta llamada sólo afecta al contenido del registro BC, dejándolo en cero ya que lo utiliza como contador de repetición de bucle.

CALL &H0059

Esta llamada transfiere un bloque de la RAM de video a la RAM principal. HL será el puntero de inicio de bloque en VRAM; DE llevará la dirección inicial de la RAM a partir de la cual se introducirá la copia y BC será el indicador de la longitud del bloque a transferir.

Ejemplo de ejecución:

```
LD HL,6144
LD DE,50000
LD BC,768
CALL &H0059
```

Tiene el efecto de depositar a partir de la dirección 50000 de la RAM el contenido de las 768 casillas del mapa de pantalla en los modos gráficos I y II.

A la salida de esta llamada el registro BC estará a cero y el registro DE quedará incrementado en tantas veces como indique BC. En este caso DE=50000+768.

CALL &H005C

Esta entrada ocasiona el efecto contrario a la anterior. Es decir, pasa el contenido de la memoria principal a la RAM de video.

En HL estará ahora, la dirección inicial en RAM y en DE la dirección de DEstino en VRAM. El resultado será transferir el bloque apuntado por HL a partir de la dirección contenida en DE y con la longitud dada por BC (Byte Counter).

Ejemplo de ejecución:

```
LD HL,50000
LD DE,6144
LD BC,768
CALL &H005C
```

Este ejemplo es el contrario al anterior y ocasiona la restauración del archivo de pantalla memorizado mediante la llamada anterior.

A la salida de esta rutina BC debe estar a cero, como es lógico. Sin embargo, HL contendrá el valor inicial dado en DE y DE tendrá el valor inicial de HL incrementado en el contenido de BC. Esto es fácil de comprender si tenemos en cuenta que lo primero que hace esta entrada es una instrucción:

EX DE,HL

que intercambia el valor de HL con el de DE.

Por tanto, en el ejemplo anterior, a la salida de esta rutina obtendríamos:

BC = 0
DE =50000+768
HL =6144

CALL &H005F

Activa uno de los cuatro modos de pantalla según el valor del acumulador.

Es decir:

LD A,0 + CALL &H005F = SCREEN 0
LD A,1 + CALL &H005F = SCREEN 1
LD A,2 + CALL &H005F = SCREEN 2
LD A,3 + CALL &H005F = SCREEN 3

A la salida, la variable RAM SCRMOD (MODo SCReen), contendrá el valor del acumulador quedando afectados todos los registros (AF, BC, DE, HL).

Una vez vistos los puntos de entrada más importantes, pasamos a ofrecerle el listado de un juego de caracteres distinto al de la ROM que nos permita realizar unos ejemplos de construcción, archivo y manejo de pantallas en Screen 1.

```
10 FORI=0TO48:READ A$
20 POKE 58950!+I,VAL("&H"+A$)
30 NEXT
40 DATA 21,78,E6,01,78,00,11,00
50 DATA 00,CD,5C,00,21,F0,E6,01
60 DATA E0,03,11,00,04,CD,5C,00
70 DATA 21,78,E6,01,78,00,11,00
80 DATA 00,CD,5C,00,21,F0,E6,01
90 DATA E0,03,11,00,0C,CD,5C,00
100 DATA C9
```


Una vez tecleado y ejecutado mediante RUN, haga NEW y teclee todos los datos listados a continuación. La misión del listado anterior es, simplemente, colocar en el sitio adecuado las rutinas en CM encargadas de pasar a VRAM los datos que vienen a continuación.

Juego de caracteres para ejemplos de pantalla

```
10 SCREEN1,2
20 FORI=0TO1111
30 READ A$
40 A=VAL("&H"+A$)
50 POKE&HE678+I,A
60 N=N+A
70 NEXT
80 IFN<>123637! THEN PRINT"ERROR EN DATA":S
TOP
90 DATA FF,10,10,10,FF,80,80,80
100 DATA C3,3C,18,00,FF,80,80,80
110 DATA FF,10,10,10,E0,80,80,80
120 DATA 80,80,80,80,80,80,80,80
130 DATA 01,01,01,01,0F,01,01,80
140 DATA 00,7E,66,66,66,66,66,00
150 DATA 91,92,94,98,90,A0,C0,80
160 DATA 91,92,94,98,A2,C2,82,82
170 DATA 00,7E,7E,7E,7E,7E,7E,00
180 DATA 38,7C,FE,FE,FE,7C,38,00
190 DATA C0,C0,C0,C0,C0,C0,C0,C0
200 DATA 00,00,00,0C,0C,00,00,00
210 DATA E0,E0,E0,E8,EF,E8,E0,E0
220 DATA C0,C0,C0,CC,CC,C0,C0,C0
230 DATA 29,D6,92,92,B2,9E,92,92
240 DATA 80,C0,E0,F0,F8,FC,FE,FF
250 DATA 01,03,07,0F,1F,3F,7F,FF
260 DATA FF,FE,FC,F8,F0,E0,C0,80
270 DATA 7F,3F,0F,07,03,01,00,00
280 DATA FF,FF,FF,FF,FF,FF,3F,1F
290 DATA FF,FF,FF,7F,1F,0F,03,01
300 DATA FF,FF,FF,FF,FF,FF,FF,FF
310 DATA 00,00,00,00,00,00,00,00
320 DATA FF,7F,3F,1F,0F,07,03,01
330 DATA FF,FE,FC,F8,F0,E0,C0,80
340 DATA 01,01,01,01,01,01,01,01
350 DATA F0,FC,7E,3F,0F,07,03,01
360 DATA 00,00,00,80,C0,F0,FC,FE
```

```
370 DATA AB,AB,6B,6B,2B,2B,2B,2B
380 DATA 2B,1A,1A,0B,0B,0B,05,03
390 DATA 55,55,56,56,54,54,54,54
400 DATA AC,A8,A8,B0,B0,B0,A0,C0
410 DATA AB,AB,AB,AB,AB,AB,AB,AB
420 DATA C3,C3,C3,C3,C3,3C,3C,10
430 DATA 80,80,80,80,80,80,80,80
440 DATA 10,38,28,38,10,10,28,38
450 DATA FF,FF,FF,DB,C3,C3,DB,FF
460 DATA 07,7C,42,59,5D,45,39,02
470 DATA E0,3C,42,9A,BA,A2,9C,40
480 DATA FF,FF,FF,E0,E0,E0,E0,E0
490 DATA E0,E0,E0,E0,E0,FF,FF,FF
500 DATA FF,FF,FF,07,07,07,07,07
510 DATA 07,07,07,07,07,FF,FF,FF
520 DATA FF,FF,FF,00,00,00,00,00
530 DATA 00,00,00,00,00,FF,FF,FF
540 DATA E0,E0,E0,E0,E0,E0,E0,E0
550 DATA 07,07,07,07,07,07,07,07
560 DATA 00,00,00,00,00,00,00,00
570 DATA 20,60,E0,E0,E0,E0,E0,E0
580 DATA E0,E0,E0,E0,E0,E0,E0,E0
590 DATA E0,E0,E0,E0,E0,E0,60,20
600 DATA 1F,3F,7F,FF,00,00,00,00
610 DATA FF,FF,FF,FF,FF,00,00,00
620 DATA F8,FC,FE,FF,00,00,00,00
630 DATA FF,00,18,00,FF,00,00,00
640 DATA E7,E7,7E,3C,3C,18,18,00
650 DATA 00,00,00,00,00,00,00,00
660 DATA FC,FE,FF,FF,FF,FE,F8,F0
670 DATA FF,FF,FF,3F,1F,1F,3F,7F
680 DATA 80,C0,E0,E0,C0,C0,80,80
690 DATA 01,03,07,07,03,03,01,01
700 DATA FF,7F,3F,1F,0F,07,03,01
710 DATA FF,FE,FC,F8,F0,E0,C0,80
720 DATA 43,AC,58,30,30,60,C0,C0
730 DATA C2,35,1A,0C,0C,06,03,03
740 DATA C0,C0,80,80,80,80,C0,C0
750 DATA 03,03,01,01,01,01,03,03
760 DATA C0,C0,60,30,30,58,AC,43
770 DATA 03,03,06,0C,0C,32,65,C2
780 DATA 00,00,00,00,00,00,00,00
790 DATA C3,3C,18,00,00,00,00,00
800 DATA FF,7F,3F,1F,0F,07,03,01
```



```
810 DATA FF,FE,FC,F8,F0,E0,C0,80
820 DATA 80,C0,E0,F0,F8,FC,FE,FF
830 DATA 00,07,1F,3F,1F,3F,3F,1F
840 DATA 00,E0,F8,FC,F8,FC,FC,F8
850 DATA 00,07,03,03,0F,0F,0F,07
860 DATA 00,E0,C0,C0,E0,F0,F0,E0
870 DATA 00,00,00,E0,F0,F7,F7,F7
880 DATA 01,03,07,0F,1F,3F,7F,FF
890 DATA 80,C0,E0,F0,F8,FC,FE,FF
900 DATA 49,94,20,48,90,20,40,80
910 DATA 49,14,22,09,04,02,01,00
920 DATA 00,00,00,1F,7F,7F,7F,3F
930 DATA 00,00,00,F0,FC,FF,FC,F8
940 DATA 00,00,00,00,00,00,00,00
950 DATA 00,00,00,00,00,00,00,00
960 DATA FF,FF,FF,FF,FF,FF,FF,FF
970 DATA FF,FE,FC,F8,F0,E0,C0,80
980 DATA 00,FF,FF,FF,FF,FF,FF,FF
990 DATA 01,03,07,0F,1F,3F,7F,FF
1000 DATA FF,7F,3F,1F,0F,07,03,00
1010 DATA 80,C0,E0,F0,F8,FC,FE,FF
1020 DATA 00,00,00,00,00,00,00,00
1030 DATA 00,00,00,00,00,00,00,00
1040 DATA 9D,EA,88,88,88,88,88,FF
1050 DATA 49,94,22,49,94,22,49,94
1060 DATA 00,80,00,40,90,20,48,94
1070 DATA 01,00,02,09,14,22,49,94
1080 DATA 22,49,94,22,49,94,22,49
1090 DATA 94,22,49,94,22,49,94,22
1100 DATA 00,00,00,00,00,00,00,00
1110 DATA 00,00,00,00,00,00,00,00
1120 DATA 28,D6,92,92,92,92,92,FF
1130 DATA 10,20,50,0A,54,22,50,08
1140 DATA 80,80,80,80,80,80,80,80
1150 DATA 21,41,A1,15,A9,45,51,11
1160 DATA 00,00,00,00,00,00,00,00
1170 DATA 00,00,00,00,00,00,00,00
1180 DATA 10,18,9C,CD,DD,FF,CA,27
1190 DATA 00,00,13,6F,E6,FF,C5,E3
1200 DATA 00,00,01,03,07,1F,3F,FF
1210 DATA 00,00,FF,FF,FF,FF,FF,FF
1220 DATA 00,00,80,C0,E0,F8,FC,FF
1230 DATA 00,00,00,50,2B,51,29,01
1240 DATA 00,00,00,14,CA,94,8A,80
1250 DATA 3C,7E,FE,FF,FF,FF,FC,38
```

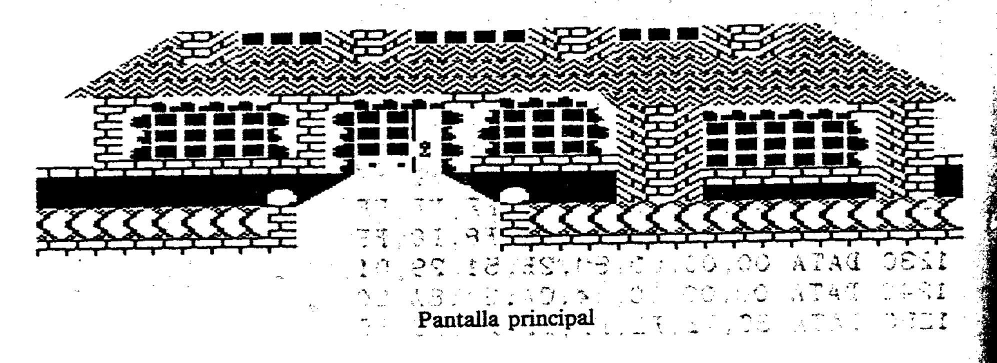
```
1260 DATA 81,C3,E7,EF,F7,F7,E3,C3
1270 DATA 00,00,00,00,00,00,00,00
1280 DATA 11,22,44,88,90,A0,C0,80
1290 DATA 91,A2,C4,88,11,32,54,98
1300 DATA 89,45,23,11,88,4C,2A,19
1310 DATA 88,44,22,11,09,05,03,01
1320 DATA 00,7E,7E,7E,7E,7E,7E,00
1330 DATA 29,D6,92,92,B2,9E,92,10
1340 DATA 1F,0F,07,07,07,0F,1F,3F
1350 DATA F0,E0,C0,C0,E0,E0,F0,FC
1360 DATA C0,C0,C0,CC,FF,80,80,FF
1370 DATA 00,00,00,66,FF,01,01,FF
1380 DATA 03,03,03,33,FF,01,01,FF
1390 DATA 00,C0,C0,C0,FF,F8,F8,FF
1400 DATA 00,00,18,3C,FF,00,00,FF
1410 DATA 00,03,03,03,FF,1F,1F,FF
1420 DATA 7F,3F,1F,07,03,01,00,00
1430 DATA FE,FC,F8,E0,C0,80,00,00
1440 DATA 03,03,07,07,0F,0F,1F,1F
1450 DATA C0,C0,E0,E0,F0,F0,F8,F8
1460 DATA 1F,1F,3F,3F,3F,7F,7F,7F
1470 DATA F8,F8,FC,FC,FC,FE,FE,FE
```

Ejecute el programa y compruebe si no ha habido errores. Tenga especial cuidado en la introducción de los datos puesto que si hay algún error en los datos, lo notaremos posteriormente como defectos en las pantallas gráficas.

Si todo ha ido bien, haga NEW y prepare una cinta para grabar los datos y la rutina listada anteriormente. Sálvelos mediante BSAVE"DATOS",58950,60150.

De ahora en adelante, para cargar la VRAM con el nuevo juego de caracteres gráficos introducido anteriormente utilizaremos:

SCREEN1:BLOAD"DATOS",R



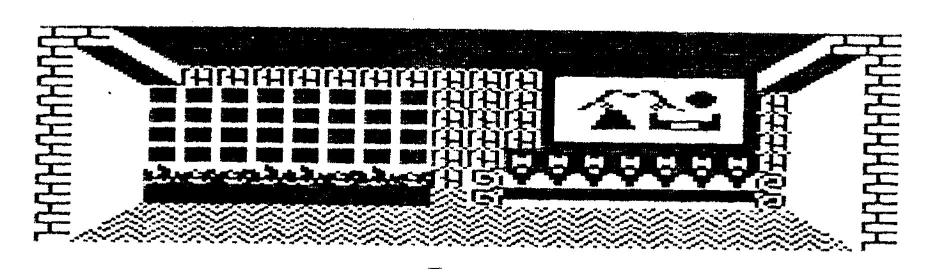
Pantalla principal


```
10 FORI=0TO447
20 READ A$
30 A=VAL("&H"+A$)
40 POKE 59151!+I,A
50 N=N+A
60 NEXT
70 IF N<> 63408! THEN PRINT"ERROR EN DATA":
STOP
80 DATA DC,DC,DC,DC,D3,00,E9,08,08,08,EA,00
,E9,08,08,08,08,EA,00,E9,08,08,08,EA,00,E9,
D1,D2,DC,DC,DC,DC
90 DATA DC,DC,DC,D3,D1,00,E9,D1,D1,D1,EA,00
,E9,D1,D1,D1,D1,EA,00,E9,D1,D1,D1,EA,00,E9,
D1,D1,D2,DC,DC,DC
100 DATA DC,DC,D3,D1,D1,D1,D1,D1,D1,D1,D1,D
1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1
,D1,D1,D1,D2,DC,DC
110 DATA DC,D3,D1,D1,D1,D1,D1,D1,D1,D1,D1,D
1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1,D1
,D1,D1,D1,D1,D2,DC
120 DATA DC,DC,00,00,BF,BF,BF,BF,00,00,00,B
F,BF,BF,00,00,BF,BF,BF,C3,D1,D1,D1,D1,D1,D1
,D1,D1,D1,D1,D1,D2
130 DATA DC,DC,00,BB,08,08,08,08,BC,00,BB,0
8,08,93,BC,BD,08,08,08,BE,EA,00,E9,BF,BF,BF
,BF,BF,00,EA,00,DC
140 DATA DC,DC,00,BD,08,08,08,08,BE,00,BD,0
8,08,93,BE,BB,08,08,08,BC,EA,00,E9,08,08,08
,08,08,BE,EA,00,DC
150 DATA DC,DC,00,BB,08,08,08,08,BC,00,BB,0
8,08,94,BC,BD,08,08,08,BE,EA,00,E9,08,08,08
,08,08,BC,EA,00,DC
160 DATA 12,12,00,00,00,00,00,00,00,00,BD,0
B,0B,93,BE,00,00,00,00,00,EA,00,E9,08,08,08
,08,08,BE,EA,00,12
170 DATA C8,C8,C8,C8,C8,C8,C8,C8,C8,C8,C9,C
E,CE,CE,CC,C8,C8,C8,C8,C8,EA,00,E9,00,00,00
,00,00,00,EA,00,C8
180 DATA C8,C8,C8,C8,C8,C8,C8,C8,E5,C9,CE,C
E,CE,CE,CE,CC,E5,C8,C8,C8,EA,00,E9,C8,C8,C8
,C8,C8,C8,EA,00,C8
190 DATA B0,B0,B0,B0,B0,B0,B0,B0,00,CE,CE,C
E,CE,CE,CE,CE,00,B0,B0,B0,B0,B0,B0,B0,B0,B0
,B0,B0,B0,B0,B0,B0
200 DATA B4,B4,B4,B4,B4,B4,B4,B4,00,CE,CE,C
```

```
E,CE,CE,CE,CE,00,B4,B4,B4,B4,B4,B4,B4,B4,B4
,B4,B4,B4,B4,B4,B4
210 DATA 00,00,00,00,00,00,00,00,00,CE,CE,C
E,CE,CE,CE,CE,00,00,00,00,00,00,00,00,00,00
,00,00,00,00,00,00
```

Ejecute el programa y compruebe como siempre, si no ha habido errores en la introducción de los datos.

Descanse un ratito si lo cree conveniente y prepárese para introducir los datos de la segunda pantalla, borrando el programa anterior, ya que salvaremos juntas las dos pantallas.



El salón

```
10 FORI=0TO263
20 READ A$
30 A=VAL("&H"+A$)
40 POKE58887!+I,A
50 N=N+A
60 NEXT
70 IF N<> 38501! THEN PRINT"ERROR EN DATA":
STOP
80 DATA 00,00,B8,86,86,86,86,86
90 DATA 86,86,86,86,86,86,86,86
100 DATA 86,86,86,86,86,B9,00,00
110 DATA 00,88,80,B8,86,86,86,86
120 DATA 86,86,86,86,86,86,86,86
130 DATA 86,86,86,86,B9,81,89,00
140 DATA 00,87,88,80,0E,0E,0E,0E
150 DATA 0E,0E,0E,0E,0E,0E,98,9C
160 DATA 9C,9C,9C,9A,81,89,87,00
170 DATA 00,87,87,EC,EC,EC,EC,EC
180 DATA EC,EC,EC,0E,0E,0E,9E,B5
```



```
190 DATA B7,B4,09,9F,0E,87,87,00
200 DATA 00,87,87,EC,EC,EC,EC,EC
210 DATA EC,EC,EC,0E,0E,0E,9E,F8
220 DATA F9,F3,F5,9F,0E,87,87,00
230 DATA 00,87,87,EC,EC,EC,EC,EC
240 DATA EC,EC,EC,0E,0E,0E,99,9D
250 DATA 9D,9D,9D,9B,0E,87,87,00
260 DATA 00,87,87,EC,EC,EC,EC,EC
270 DATA EC,EC,EC,0E,0E,95,95,95
280 DATA 95,95,95,95,0E,87,87,00
290 DATA 00,87,87,DE,DF,DF,DE,DE
300 DATA DF,DE,DF,0E,96,A8,A8,A8
310 DATA A8,A8,A8,A8,97,87,87,00
320 DATA 00,87,87,C8,C8,C8,C8,C8
330 DATA C8,C8,C8,D5,96,A5,A5,A5
340 DATA A5,A5,A5,A5,97,87,87,00
350 DATA 00,87,D3,D1,D1,D1,D1,D1
360 DATA D1,D1,D1,D1,D1,D1,D1,D1
370 DATA D1,D1,D1,D1,D1,D2,87,00
380 DATA 00,D3,D4,D4,D4,D4,D4,D4
390 DATA D4,D4,D4,D4,D4,D4,D4,D4
400 DATA D4,D4,D4,D4,D4,D4,D2,00
```

Ejecute el programa y salve las dos pantallas juntas mediante:

BSAVE"SCREEN",58887,60000

Antes de continuar cansando al lector con nuevos listados, vamos a repasar lo que hemos hecho:

- Se ha grabado en el programa DATOS un nuevo juego de caracteres que nos permita realizar muchas pantallas. Su finalidad es, simplemente, cargar la VRAM con lo cual su posterior destrucción en la RAM no nos afecta.
- Se han grabado dos ejemplos de pantallas para el modo gráfico I.
- Por tanto sólo nos queda introducir las rutinas CM que nos permitan traer a la VRAM, adecuadamente, los datos.

En efecto, teclee lo siguiente:

```
10 FORI=0TO67:READ A$
20 POKE&HF500+I,VAL("&H"+A$):NEXT
30 FORI=0TO63:READ A
40 POKE61400!+I,A:NEXT
```

```
50 DEFUSR=&HF500:DEFUSR1=&HF510
60 POKE62336!,1:A=USR(0)
70 A$=INKEY$:IFA$<>" "THEN70
80 CLS:POKE62336!,2:A=USR1(0)
90 A$=INKEY$:IFA$<>" "THEN90
100 POKE62336!,2:A=USR1(0):GOTO60
110 DATA 21,0F,E7,11,00,18,01,C0
120 DATA 01,CD,5C,00,CD,2E,F5,C9
130 DATA 11,07,E6,21,FA,17,06,0B
140 DATA C5,01,08,00,09,06,18,1A
150 DATA CD,4D,00,13,23,10,F8,C1
160 DATA 10,EE,CD,2E,F5,C9,3A,80
170 DATA F3,01,20,00,21,D8,EF,3D
180 DATA 28,03,09,18,FA,11,00,20
190 DATA CD,5C,00,C9
200 '.**** DATOS DE COLORES ****
210 DATA 24,161,245,,,,,,,,,,,,,,,,161,,,,6
0,246,22,43,21,5,242,182,,
220 DATA 10,21,,,,,,,,,,,,,,,39,39,182,245,
182,182,53,39,,128,7,206,244,228,69,149
```

Este listado se puede salvar en cinta mediante CSAVE"Ejemplo". Su misión es traer dos pantallas distintas y en diferentes formatos mediante la pulsación de la barra espaciadora. Ha realizado un programa en CM. Todo consiste en salvar adecuadamente en cinta diversas longitudes de memoria. El listado BASIC dirige las dos rutinas.

La primera función USR trae una pantalla de 32×14 octetos y la segunda, otra de 24×11. Además hemos tenido que crear un archivo de 32 datos para cada pantalla, encargado de memorizar los colores de ambas.

Observe cómo se utiliza la casilla límite 62336 como variable de color. En efecto, los 32 primeros datos corresponden a la primera pantalla y los 32 siguientes a la segunda. Es el programador, por tanto, el que decide la distribución o paginado de las pantallas.

En este caso hemos asignado un 1 a la pantalla de 32×14 cuadratines y un 2 a la 24×11 en el archivo de datos de colores. En modo gráfico I es relativamente fácil y económico memorizar íntegramente la tabla de colores. La limitación es, como ya sabe, poder tener solamente dos colores (tinta y fondo) para ocho caracteres cada vez. En Screen 2 es necesaria una cantidad enorme de datos en la tabla de color para cada carácter; del orden de 64 veces más comparativamente, teniendo además tres ventanas independientes.

En el apartado Screen 1 en modo gráfico le propondremos un ejemplo de lo dicho anteriormente. Para ello se recurre a una rutina de "comprensión" de datos que permita llenar la tabla de color correspondiente a las dos ventanas superiores de la pantalla. A efectos de trabajo en CM es mejor pensar en términos de Screen 1 en modo gráfico, que de Screen 2 ya que la técnica de programación es muy similar.

Estudie detenidamente las rutinas USR(0) y USR1(0) puesto que variando desde el BASIC la dirección contenida en HL es posible paginar adecuadamente la RAM para llegar a obtener de 20 a 40 pantallas distintas dependiendo de su longitud.

No vamos a explicar aquí el listado Assembler de las rutinas empleadas puesto que esperamos que sea usted el interesado en perfeccionarlas, ya que han sido hechas expresamente para el ejemplo.

El juego de caracteres que le hemos dado se adecua muy bien para realizar pantallas de viviendas e interiores. Con él puede diseñar más pantallas e incluir un posible guión para un programa ...

En el siguiente apartado hemos realizado unas pequeñas herramientas en CM que quizás le sean muy útiles. Por nuestra parte esperamos que el ejemplo realizado le haya gustado y le anime a realizar sus propias creaciones.

3. Efectos especiales

En este apartado vamos a dar algunas ideas útiles para incluir en sus programas BASIC que le permitan realizar efectos especiales de una manera rápida mediante pequeñas rutinas en CM.

La intención que perseguimos con ello es esencialmente didáctica. Por tanto aunque se puedan utilizar directamente es conveniente que las estudie e intente comprenderlas y adaptarlas a sus necesidades. Con un poco de práctica también podrá crear sus propias rutinas.

Rutina de sustitución de caracteres

```
                   ;RUTINA PARA SUSTITUIR CARACTERES
                   ;REUBICABLE
                   ;(C) J&J. ASIN
F500                            ORG   #F500
F500  210018                    LD    HL,6144
F503  0656                      LD    B,"V"
F505  0E4E                      LD    C,"N"
F507  110003                    LD    DE,768
F50A  CD4A00       LECTURA:     CALL  #004A
F50D  B8                        CP    B
F50E  2004                      JR    NZ,ESCRIT
F510  79                        LD    A,C
F511  CD4D00                    CALL  #004D
F514  23           ESCRIT:      INC   HL
F515  1B                        DEC   DE
F516  7B                        LD    A,E
F517  B2                        OR    D
F518  20F0                      JR    NZ,LECTURA
F51A  C9                        RET
```

En la línea 10 se apunta en HL la dirección de comienzo del mapa de pantalla en los modos gráficos I y II (6144).

En la línea 20 se debe cargar en el registro B el número de código del carácter que queremos sustituir. Lo hemos indicado mediante LD B,"V" para referirnos al carácter a cambiar (Viejo). La casilla de memoria donde deberemos "pokear" será la inicial donde se ubique la rutina incrementada en 4.

En la línea 30 deberemos introducir en el registro C el número de código del carácter nuevo que sustituya al anterior (dirección inicial+6).

En la línea 40 se introduce en DE el número 768, indicativo de contador bucle, ya que es el número total de casillas de archivo de pantalla en los modos gráficos I y II.

La llamada BIOS de la línea 50 ya es conocida por usted. Su misión es meter en el acumulador el contenido de la casilla de la VRAM a la que apunta HL.

En las líneas 60-90 se compara si el dato leído en A es igual al introducido en B. Si no es así, pasará a la línea 100 incrementando HL (casilla siguiente), y decrementando el registro DE, contador de bucle. En caso afirmativo se sustituirá en A el número de carácter que contenga C, siendo a continuación el CALL BIOS que usted ya conoce el encargado de sustituirlo en la casilla de la VRAM investigada.

Las líneas 120-140 son el detector de bucle propiamente dicho. Obsérvelas detenidamente ya que es la técnica más rápida de conocer si un registro doble ha llegado a ser 0. En efecto, se carga en A la parte baja llevada por E y se hace una operación lógica OR con la parte alta contenida en D. Sólo si ambos registros contienen cero, se posicionará el flag correspondiente. En ese caso la línea 150 hará que retorne a BASIC, de lo contrario volverá a iniciar el proceso en la etiqueta LECTURA, siendo investigada la siguiente casilla del mapa de pantalla.

Rutina para intercambio de colores

```
                  1  ;RUTINA PARA INTERCAMBIO
                  2  ;DE COLORES FONDO X TINTA
                  3  ;ADAPTABLES A LA INVERSA
                  4  ;REUBICABLE
                  5  ;(C) J&J. ASIN
F500              6            ORG   #F500
F500  210020     10            LD    HL,8192
F503  0E00       20            LD    C,0
F505  0646       30            LD    B,"F"
F507  3E20       40            LD    A,32
F509  F5         50  NUEVO:    PUSH  AF
F50A  CD4A00     60            CALL  #004A
F50D  E60F       70            AND   15
F50F  B8         80            CP    B
F510  201F       90            JR    NZ,DAT
F512  CD4A00    100            CALL  #004A
F515  E60F      110            AND   15
```



```
F517  5F        120            LD    E,A
F518  CD4A00    130            CALL  #004A
F51B  CB0F      140            RRC   A
F51D  CB0F      150            RRC   A
F51F  CB0F      160            RRC   A
F521  CB0F      170            RRC   A
F523  E60F      180            AND   15
F525  CB03      190            RLC   E
F527  CB03      200            RLC   E
F529  CB03      210            RLC   E
F52B  CB03      220            RLC   E
F52D  B3        230            O     E
F52E  CD4D00    240            CALL  #004D
F531  23        250  DAT:      INC   HL
F532  F1        260            POP   AF
F533  3D        270            DEC   A
F534  20D3      280            JR    NZ,NUEVO
F536  C9        290            RET
```

Observe la gran similitud con la rutina anterior. En este caso HL debe contener la dirección de inicio de la tabla de colores que como ya sabe es 8192, a no ser que la hayamos situado en otro lugar mediante la función VDP.

En la línea 40 se carga en el acumulador el valor 32, que es el número de casillas encargadas de memorizar el color de tinta y fondo, en bloques de 8 en 8, de los 256 caracteres posibles. Esto es válido solamente para Screen 1. Le proponemos que intente adaptarla al modo gráfico II, para lo cual podría utilizar el registro DE cargándolo con 6144 ...

Observe las instrucciones PUSH AF y POP AF (líneas 50 y 260 respectivamente) para salvaguardar el acumulador momentáneamente.

En la línea 70 y mediante la operación lógica AND 15 (&B00001111) se eliminan los cuatro Bits más significativos encargados del color de tinta, preservando los cuatro Bits más significativos encargados del color de fondo.

Las líneas 80-130 detectan si el valor introducido en el registro B coincide con el dato leído en A. A continuación se utilizan los registros E y A para rotarlos e invertir los Bits de manera que los cuatro Bits menos significativos (color de fondo) pasen a ser los cuatro más significativos (color de tinta) y a la inversa. ¿Qué le parece?

Para que esta rutina funcione, sólo deberá introducir en el registro B de la línea 30 un número indicativo del código de color de fondo. La rutina analizará toda la tabla de colores intercambiando el color de tinta por el de fondo que hayamos indicado.

Rutina para borrar áreas especificadas

```
                10  ;RUTINA PARA BORRAR AREAS
                20  ;ESPECIFICADAS. REUBICABLE
                30  ;(C) J&J. ASIN
F500            35             ORG   #F500
F500  210000    40             LD    HL,0
F503  012000    50             LD    BC,32
F506  3E46      60             LD    A,"F"
F508  3D        70  FILA:      DEC   A
F509  2803      80             JR    Z,COLUMNA
F50B  09        90             ADD   HL,BC
F50C  18FA     100             JR    FILA
F50E  110000   110  COLUMNA:   LD    DE,0
F511  3E43     120             LD    A,"C"
F513  3D       130             DEC   A
F514  5F       140             LD    E,A
F515  19       150             ADD   HL,DE
F516  110018   160             LD    DE,6144
F519  19       170             ADD   HL,DE
F51A  3E41     180             LD    A,"A"
F51C  F5       190  BUCLE:     PUSH  AF
F51D  3E57     200             LD    A,"W"
F51F  E5       210             PUSH  HL
F520  F5       220  BUCLE2:    PUSH  AF
F521  3E20     230             LD    A,32
F523  CD4D00   240             CALL  #004D
F526  23       250             INC   HL
F527  F1       260             POP   AF
F528  3D       270             DEC   A
F529  20F5     280             JR    NZ,BUCLE2
F52B  E1       290             POP   HL
F52C  09       300             ADD   HL,BC
F52D  F1       310             POP   AF
F52E  3D       320             DEC   A
F52F  20EB     330             JR    NZ,BUCLE
F531  C9       340             RET
```

Las líneas 40-170 realizan una simple suma en HL para averiguar la cuadrícula del archivo de pantalla a partir de la cual se borrará determinada zona de acuerdo a los parámetros que contenga el acumulador en las líneas 180 (altura) y 200 (width o anchura).

Observe cómo a la salida, HL será igual a 6144 + 32 * fila + columna, fórmula que ya conoce y que aquí se realiza mediante los registros HL, BC, DE y A:

HL=0+BC*A+E+DE(=6144)

Multiplicar en CM se reduce a sumar repetidas veces, pero el objetivo queda perfectamente cumplido en la línea 170.

Las líneas 180-340 realizan el borrado o relleno de la zona de pantalla elegida con el código del carácter de espaciado (32) de la línea 230.

Adaptar esta rutina a sus necesidades, como puede comprobar es relativamente fácil. Puede incluso borrar o rellenar parte de determinada zona de la pantalla con el carácter deseado (0-255) mediante el oportuno cambio en la línea 230 que pasaría a ser LD A,N donde N será el número comprendido entre 0 y 255 que indique el código de carácter seleccionado.

Rutina para pasar datos de VRAM a RAM

```
                 1  ; RUTINA PARA PASAR VRAM A RAM
                 2  ; FACIL PROCESO INVERSO
                 3  ; REUBICABLE
                 4  ; (C) J&J. ASIN
F500             5             ORG   #F500
F500  210000    10             LD    HL,0
F503  012000    20             LD    BC,32
F506  3E46      30             LD    A,"F"
F508  3D        40  FILA:      DEC   A
F509  2803      50             JR    Z,COLUMNA:
F50B  09        60             ADD   HL,BC
F50C  18FA      70             JR    FILA
F50E  110000    80  COLUMNA:   LD    DE,0
F511  3E43      90             LD    A,"C"
F513  3D       100             DEC   A
F514  5F       110             LD    E,A
F515  19       120             ADD   HL,DE
F516  110018   130             LD    DE,6144
F519  19       140             ADD   HL,DE
F51A  1148EE   150             LD    DE,61000
F51D  00       151             NOP
F51E  3E41     160             LD    A,"A"
F520  F5       170  BUCLE:     PUSH  AF
F521  C5       180             PUSH  BC
F522  D5       190             PUSH  DE
F523  E5       200             PUSH  HL
F524  010000   210             LD    BC,0
F527  3E57     220             LD    A,"W"
F529  4F       230             LD    C,A
F52A  CD5900   240             CALL  #0059
F52D  E1       250             POP   HL
F52E  D1       260             POP   DE
```

```
F52F  C1      270         POP   BC
F530  09      280         ADD   HL,BC
F531  EB      290         EX    DE,HL
F532  09      300         ADD   HL,BC
F533  EB      310         EX    DE,HL
F534  F1      320         POP   AF
F535  3D      330         DEC   A
F536  20E8    340         JR    NZ,BUCLE
F538  C9      350         RET
```

La mecánica de esta rutina es parecida a la anteriormente vista. Observe cómo las líneas 10-140 realizan el mismo cálculo con el fin de obtener la casilla de pantalla seleccionada.

Los valores de altura y anchura de la zona elegida para memorizar en RAM deben introducirse en las líneas 160 y 220. La línea 150 carga DE con 61000, que ha sido la casilla elegida por nosotros a partir de la cual se hará la copia. Finalmente, dése cuenta del "apilamiento" de todos los registros que intervienen en el cálculo antes del CALL de la línea 240 y su posterior "desapilamiento" en orden inverso según la regla LIFO que usted ya conoce.

Esta rutina le será muy útil, adaptándola a sus necesidades. Así, por ejemplo, en la línea 150 deberá introducir la casilla de memoria a partir de la cual se desea obtener la copia, dándola siempre en el orden parte baja, parte alta.

Haga pruebas de memorización de determinadas zonas hasta que se familiarice con los datos que debe introducir en las líneas 30, 90, 160 y 220. Observe cómo las líneas 30 y 90 son similares a un LOCATE del BASIC pero a la inversa. Es decir LOCATE 31,23 sería aquí 24,32. Por tanto, en la línea 30 el acumulador deberá contener un número comprendido entre 1 y 24 y en la línea 90 entre 1 y 32.

Esta rutina, por sí sola no sirve para comprobar si la memorización de pantalla ha sido correcta. Evidentemente, falta una rutina que haga lo contrario. Si ha entendido el funcionamiento de ésta, le rogaríamos que pensase sobre el papel los cambios que debería hacer para lograr el efecto inverso. Si cree que no lo sabe continúe leyendo la explicación que sigue, pero intente ya desde ahora con ilusión animarse a realizar sus propias rutinas. Si no posee Ensamblador, utilice la tabla del apéndice para ensamblar a mano; y sobre todo no se desanime nunca; piense que aunque al principio le cueste más tiempo el hacer el equivalente en CM al BASIC, el resultado será, a la larga, más gratificante.

Adecuar la rutina para que realice el efecto inverso es sencillo ya que el cálculo principal está hecho. En efecto, sólo deberemos cambiar la llamada de la línea 240 por CALL &H005C que a efectos de memoria quedaría reducido a sustituir &H59 por &H5C. Previamente, sin embargo hay que modificar las líneas 150 y 151 de la siguiente forma:

```
150 EX DE,HL(&HEB)
151 LD HL,61000
```


El cambio de la línea 151 sólo afecta en memoria a la sustitución del código &H11 por &H21.

Rutina de SCROLL vertical

```
              10  ;RUTINA DESPLAZAMIENTO DE
              20  ;PANTALLA HACIA ARRIBA
              30  ;FACIL MODIFICACIÓN INVERSA
              40  ;REUBICABLE
              50  ;(C) J&J. ASIN
F500          55              ORG   #F500
F500  210000  60              LD    HL,0
F503  012000  70              LD    BC,32
F506  3E46    80              LD    A,"F"
F508  3D      90  FILA:       DEC   A
F509  2803   100              JR    Z,COLUMNA
F50B  09     110              ADD   HL,BC
F50C  18FA   120              JR    FILA
F50E  110000 130  COLUMNA:    LD    DE,0
F511  3E43   140              LD    A,"C"
F513  3D     150              DEC   A
F514  5F     160              LD    E,A
F515  19     170              ADD   HL,DE
F516  110018 180              LD    DE,6144
F519  19     190              ADD   HL,DE
F51A  E5     200              PUSH  HL
F51B  D1     210              POP   DE
F51C  3E57   220              LD    A,"W"
F51E  F5     230  BUCLE:      PUSH  AF
F51F  CD4A00 240              CALL  #004A
F522  329EF2 250              LD    (62110),A
F525  3E48   260              LD    A,"H"
F527  3D     270              DEC   A
F528  F5     280  BUCLE2:     PUSH  AF
F529  09     290              ADD   HL,BC
F52A  00     300              NOP
F52B  00     310              NOP
F52C  CD4A00 320              CALL  #004A
F52F  ED42   330              SBC   HL,BC
F531  CD4D00 340              CALL  #004D
F534  09     350              ADD   HL,BC
F535  00     360              NOP
F536  00     370              NOP
F537  F1     380              POP   AF
```

```
F538  3D       390        DEC   A
F539  20ED     400        JR    NZ,BUCLE2
F53B  3A9EF2   410        LD    A,(62110)
F53E  CD4D00   420        CALL  #004D
F541  13       430        INC   DE
F542  D5       440        PUSH  DE
F543  E1       450        POP   HL
F544  F1       460        POP   AF
F545  3D       470        DEC   A
F546  20D6     480        JR    NZ,BUCLE
F548  C9       490        RET
```

El algoritmo de las líneas 60-190 ya es de sobra conocido y como puede ver, es muy útil en las rutinas que llevamos vistas.

La misión de la siguiente rutina es hacer un Scroll hacia arriba sin demasiadas complicaciones, de tal manera que lo que desaparezca de pantalla por arriba, volverá a aparecer por debajo de la misma.

Observe el truco de las líneas 200-210. Se "apila" HL y se "desapila" acto seguido pero no en HL, como marca la norma sino en DE; con lo cual se logra transferir el dato de HL a DE directamente, ya que se carace de la instrucción LD DE,HL. Un truco interesante ¿verdad?

El resto es sencillo. Se va estudiando la VRAM en dos bucles controlados por el acumulador de acuerdo a los parámetros anchura-altura que introduzca en las líneas 220 y 260 respectivamente. Se utiliza una casilla de memoria, 62110 como almacenamiento temporal del dato leído en el acumulador; retomándose el valor de esta casilla en la línea 410 para introducirlo en la VRAM.

Es posible modificar esta rutina para realizar el efecto contrario variando las siguientes líneas:

- 290 AND A (A7)
- 300 SBC HL,BC (ED 42)
- 330 ADD HL,BC (09)
- 350 AND A (A7)
- 360 SBC HL,BC (ED 42)

Los números entre paréntesis indican el código objeto de la instrucción. Deberá cambiar las casillas correspondientes según la dirección de inicio asignada. Observe cómo la única modificación ha sido cambiar oportunamente el cálculo restando en vez de sumando y a la inversa.

La introducción de un AND A previo sirve para poner a cero el flag de acarreo sin que varíe el acumulador.

Rutina de SCROLL horizontal

```
                10 ;RUTINA DESPLAZAMIENTO DE
                20 ;PANTALLA HACIA LA IZDA.
                30 ;FACIL MODIFICACION INVERSA
                40 ;REUBICABLE
                50 ;(C) J&J. ASIN
F500            55              ORG   #F500
F500  210000    60              LD    HL,0
F503  012000    70              LD    BC,32
F506  3E46      80              LD    A,"F"
F508  3D        90  FILA:       DEC   A
F509  2803     100              JR    Z,COLUMNA
F50B  09       110              ADD   HL,BC
F50C  18FA     120              JR    FILA
F50E  110000   130  COLUMNA:    LD    DE,0
F511  3E43     140              LD    A,"C"
F513  3D       150              DEC   A
F514  5F       160              LD    E,A
F515  19       170              ADD   HL,DE
F516  110018   180              LD    DE,6144
F519  19       190              ADD   HL,DE
F51A  E5       200              PUSH  HL
F51B  D1       210              POP   DE
F51C  3E57     220              LD    A,"W"
F51E  F5       230  BUCLE:      PUSH  AF
F51F  CD4A00   240              CALL  #004A
F522  329EF2   250              LD    (62110),A
F525  3E48     260              LD    A,"H"
F527  3D       270              DEC   A
F528  F5       280  BUCLE2:     PUSH  AF
F529  23       290              INC   HL
F52A  CD4A00   320              CALL  #004A
F52D  2B       330              DEC   HL
F52E  CD4D00   340              CALL  #004D
F531  23       350              INC   HL
F532  F1       380              POP   AF
F533  3D       390              DEC   A
F534  20F2     400              JR    NZ,BUCLE2
F536  3A9EF2   410              LD    A,(62110)
F539  CD4D00   420              CALL  #004D
F53C  C5       430              PUSH  BC
F53D  E1       440              POP   HL
F53E  19       450              ADD   HL,DE
F53F  E5       460              PUSH  HL
```

```
F540  D1     465    POP  DE
F541  F1     470    POP  AF
F542  3D     475    DEC  A
F543  20D9   480    JR   NZ,BUCLE
F545  C9     490    RET
```

Esta rutina utiliza un cálculo prácticamente idéntico a la rutina de Scroll vertical por lo que no entraremos en explicaciones.

A continuación, exponemos los cambios necesarios para lograr el efecto contrario:

- 290 DEC HL (2b)
- 330 INC HL (23)
- 350 DEC HL (2B)

El lector debe saber deducir, a estas alturas, la causa.

4. Screen 1 en modo gráfico

No queremos despedirnos sin demostrar la diferencia entre ambas concepciones de realización de pantallas. Le aconsejamos que practique en Screen 1 debido a la más fácil programación de la tabla de colores.

Para realizar el ejemplo siguiente deberemos cargar en Screen 1 el programa "DATOS" con autoejecución y posteriormente hacer bload"SCREEN" para ubicar en memoria las dos pantallas del ejemplo del apartado 2.

La pantalla va a ser la misma de la Mansión pero con algunos caracteres definidos con más de dos colores para notar la diferencia. El resto de tabla de color la dejaremos como estaba en Screen 1 mediante un algoritmo de compresión en CM que multiplique por 64 el dato de color a introducir.

```
10 FORI=0TO259:READ A$
20 POKE55000!+I,VAL("&H"+A$):NEXT
30 DEFUSR=&HD6EA:A=USR(0)
40 GOTO 40
50 DATA 00,00,02,00,E2,01,06,02
60 DATA FF,03,03,04,7E,05,07,06
70 DATA 01,07,06,09,21,D8,D6,C5
80 DATA 46,23,4E,23,CD,47,00,C1
```



```
90 DATA 10,F5,DD,21,9F,D7,11,00
100 DATA 20,DD,7E,00,A7,28,1B,6F
110 DATA 26,00,29,29,29,44,4D,C5
120 DATA EB,E5,DD,7E,01,CD,56,00
130 DATA E1,C1,09,EB,DD,23,DD,23
140 DATA 18,DF,21,00,20,11,7B,D7
150 DATA CD,86,D7,21,00,28,11,7B
160 DATA D7,CD,86,D7,06,05,21,D8
170 DATA 25,C5,CD,83,D7,C1,10,F9
180 DATA 06,05,21,D8,2D,C5,CD,83
190 DATA D7,C1,10,F9,21,E0,26,CD
200 DATA 91,D7,06,02,21,90,26,C5
210 DATA CD,91,D7,C1,10,F9,21,0F
220 DATA E7,11,00,18,01,C0,01,CD
230 DATA 5C,00,C9,14,14,17,19,15
240 DATA 15,17,17,F8,F6,F9,F6,F6
250 DATA F9,F6,F9,18,16,B9,B9,16
260 DATA 19,18,18,11,73,D7,06,08
270 DATA 1A,CD,4D,00,13,23,10,F8
280 DATA C9,11,6B,D7,06,08,1A,CD
290 DATA 4D,00,13,23,10,F8,C9,08
300 DATA 18,08,A1,08,F5,78,00,08
310 DATA A1,18,00,08,3C,08,F6,08
320 DATA 16,08,2B,08,15,08,05,08
330 DATA F2,08,B6,10,00,08,18,08
340 DATA A1,08,F5,78,00,08,A1,18
350 DATA 00,08,3C,08,F6,08,16,08
360 DATA 2B,08,15,08,05,08,F2,08
370 DATA B6,10,00,00
```

Para terminar este capítulo, proporcionaremos un programa de utilidad para facilitar la tarea de diseño de caracteres, Sprites y pantallas.

Manejo del programa gestor

Aunque el programa es de fácil manejo, quizá presente algunas particularidades que será bueno comentar:

- El diseño, tanto de caracteres como de Sprites se realiza sobre una parrilla de 8×8 cuadratines.
- La modificación o construcción de los caracteres se realiza introduciendo ocho números binarios en el orden que usted elija.

- El programa proporciona a la derecha de la parrilla los valores decimal y hexadecimal de cada octeto.
- Mediante [SELECT] se finaliza la construcción de un cáracter y el programa vuelve a pedir otro número de carácter, y así sucesivamente.
- Para finalizar la tarea de modificación o creación de caracteres pulse [CTRL][STOP] y siga las instrucciones del menú.
- Para diseñar Sprites de 16×16 tendrá que proceder a la creación de los cuatro bloques de 8×8 que lo conforman.
- Finalmente, debe tener en cuenta que el diseño de caracteres se realiza a partir de la dirección 2048 de la Videoram; por lo cual haciendo VDP(4)=0 pasaremos al juego normal de caracteres y con VDP(4)=1 pasaremos a la página utilizada por el Gestor, donde estará situado todo el juego de caracteres que haya diseñado.
- Los números cero de carácter o Sprite son utilizados por el programa para mostrar a tamaño natural el diseño realizado en cada momento.

Cómo salvar el programa en cinta

Debido a la bondad del programa es conveniente reservar una cinta al principio, de cara a futuros trabajos con el Gestor. Para ello, procederemos previamente al listado principal, a salvar una pequeña cabecera en formato ASCII que autoejecute el programa principal. Para ello utilizaremos técnicas aprendidas en el capítulo II. En este caso lograremos eliminar el mensaje Ok para una mayor facilidad en el uso de Gestor y autoejecutar el comando Cload.

Cabecera ASCII de carga

```
1 POKE&HFF08,&H75:POKE&HFF09,&HF9:FORI=&HF9
75TO&HF97E:READA$
2 POKEI,VAL("&H"+A$):NEXTI:POKE&HFF07,&HC3
3 DATA CD,23,73,21,7E,F9,C3,31,41,00
10 Z$="RUN"+CHR$(13):FORQ=&HFBF0TO&HFC18:PO
KEQ,0:NEXT:POKE&HF3FA,240:POKE&HF3FB,251:PO
KE&HF3FC,244:POKE&HF3FD,251:FORQ=1TOLEN(Z$)
:POKE&HFBF0+Q,ASC(MID$(Z$,Q,1)):NEXT:CLOAD
```

Una vez listada esta cabecera se procederá a su grabación en cinta mediante Save"GESTOR". A continuación se borrará la cabecera y se procederá a listar el programa principal, que salvaremos a continuación mediante CSAVE" ". Cuando se quiera cargar el programa GESTOR se utilizará simplemente el mandato RUN "CAS:".

Gestor de pantalla completa

(C) J&J ASIN GASCON.
SALAMANCA 1985.

```
**********************************
*                                *
*     PARA  "MAS ALLA DEL BASIC" *
*                                *
**********************************

10 KEYOFF:SCREEN1,0:COLOR15,4,4:WIDTH29:POK
E&HF3B1,24
20 CLEAR200,40000!:DEFINT A-Z:DEFSNG D
30 FORI=&HF500TO&HF50C
40 READA$
50 POKEI,VAL("&H"+A$)
60 NEXT
70 DATA 11,0,0,21,0,0,1,0,8,CD,59,0,C9
80 FORI=&HABD3TO&HABDF
90 READA$
100 POKEI,VAL("&H"+A$)
110 NEXT
120 DATA 21,0,18,11,e0,ab,1,0,3,cd,59,0,c9
130 DEFUSR=&HF500
140 DEFUSR1=43987!
150 VPOKE8217,20
160 STOPON:ONSTOPGOSUB760
170 FORI=0TO7:VPOKE14336+I,VPEEK(207*8+I):N
EXT
180 VPOKE 8194,113
190 VPOKE 8195,113
200 A$(3)="┌──────┬─┬──────┐"
210 A$(4)="│ NUM. │ │      │"
220 A$(5)="│XXXXXX└─┘XXXXXX│"
230 A$="┌┬┬┬┬┬┬┬┐"
240 B$="└┴┴┴┴┴┴┴┘"
250 A$(1)="│││││││││"
260 A$(2)="├┼┼┼┼┼┼┼┤"
270 LOCATE0,23:PRINTA$(3)
280 PRINTA$(4)
290 PRINTA$(5)
300 PRINTA$
310 FORI=0TO7:PRINTA$(1):IF I=7THEN 320 ELS
```

```
EPRINTA$(2):NEXT
320 PRINTB$;
325 FORI=0TO7:VPOKE6400+I*64,49+I:NEXT
330 VPOKE6314,0
340 FORI=0TO7:VPOKE6247+I,206:NEXT
350 LOCATE18,4:PRINT"HEX. DEC":
360 LOCATE 18,5:PRINT"———————":LOCATE0,0
370 FORI=0TO7:VPOKEI,0:NEXT
380 POKE&HF3B1,3:IFT<>0THEN430
390 INPUT"SPRITES O                    CARA
CTERES S/C";A$
400 IF A$="S"OR A$="s"THEN SC=14336:CA$="SP
RITE":GOTO430
410 IF A$="C"OR A$="c"THEN SC=2048:CA$="CAR
ACTER":GOTO430
420 GOSUB730:GOTO390
430 GOSUB730:PRINT"NUM.";CA$;:INPUTN
440 IFN>255 OR N<1 THEN GOSUB730:GOTO 430
450 GOSUB710
460 GOSUB730:PRINT"NUM.DE OCTETO? ";:A$=INP
UT$(1)
470 A=ASC(A$)-48
480 IFA=-24THENPOKE&HF3B1,24:CLS:T=1:VPOKE6
912,192:GOTO 200
490 IFA<1OR A>8 THEN 460
500 X=48+16*A
510 VP=6339+64*A
520 PUT SPRITE 0,(10,X),15
530 GOSUB730:LOCATE0,2:INPUT "&B=";A$
540 IF LEN(A$)>8THENBEEP:GOTO 530
550 FORI=1TO8
560 X$=MID$(A$,I,1)
570 IFVAL(X$)>1THENBEEP:GOTO 530
580 NEXT
590 VPOKE A-1,VAL("&B"+A$)
600 VPOKE SC+N*8+A-1,VAL("&B"+A$)
610 IF A$="" THEN A$="00000000"
620 IF LEN(A$)<8THEN A$=STRING$(8-LEN(A$),"
0")+A$
630 FORI=1TO8:X$=MID$(A$,I,1)
640 IF ASC(X$)=49THEN VPOKE VP,219:VP=VP+2E
LSE VPOKE VP,32:VP=VP+2
650 NEXT:VP=0
660 GOSUB680
670 GOSUB730:GOTO460
```



```
680 POKE&HF3B1,24
690 H$=HEX$(VAL("&B"+A$)):V=VAL("&H"+H$)
700 LOCATE 19,6+2*A:PRINTSTRING$(2-LEN(H$),
"0")+H$;:PRINT USING"#####";V:GOTO720
710 POKE&HF3B1,24:LOCATE 11,5:PRINTN
720 POKE&HF3B1,3:LOCATE0,0:RETURN
730 A$="":H$="":BC$="":BC$=STRING$(130,32)
740 LOCATE0,0:PRINTBC$:LOCATE0,0
750 RETURN
760 POKE&HF3B1,24:CLS
770 LOCATE10,0:PRINT"OPCIONES"
780 LOCATE10,1:PRINT"========"
790 PRINT:PRINT
800 PRINT"1.- GRABAR BLOQUE SPRITES"
810 PRINT:PRINT
820 PRINT"2.- GRABAR BLOQUE CARACTERES"
830 PRINT:PRINT
840 PRINT"3.- GESTION DE PANTALLAS"
850 PRINT:PRINT
860 PRINT"4.- VOLVER AL PROGRAMA"
870 PRINT:PRINT
880 PRINT"5.- FIN"
890 LOCATE11,22:PRINT"OPCION? ";:A$=INPUT$(
1):A=ASC(A$)-48
900 IFA<1ORA>5THEN890
910 ONAGOTO920,1080,1100,1250,1260
920 CLS:POKE&HF504,0:POKE&HF505,&H38
930 INPUT"DIR. INICIAL EN RAM";DI
940 IFDI>60000!ORDI<45000!THEN930
950 H=DI/256:L=DI-H*256
960 POKE&HF501,L:POKE&HF502,H
970 A=USR(0)
980 FORI=0TO12:POKEDI-13+I,PEEK(&HF500+I)
990 IFPEEK(&HF500+I)=33THENPOKEDI-13+I,17
1000 IFPEEK(&HF500+I)=17THENPOKEDI-13+I,33
1010 NEXT:POKEDI-3,&H5C
1020 INPUT"NOMBRE DEL PROGRAMA";A$
1030 PRINT"PULSE REC/PLAY Y DESPUES UNA TEC
LA"
1040 CLS:LOCATE5,11:PRINT"GRABANDO ";A$
1050 PRINT:PRINT"DESDE ";DI-13;" HASTA ";DI
+2048:PRINT:PRINT"AUTOEJECUCION EN ";DI-13
1060 BSAVEA$,DI-13,DI+2048,DI-13
1070 GOTO 760
```

```
1080 CLS:POKE&HF504,0:POKE&HF505,8
1090 GOTO 930
1100 CLS:POKE&HF3B1,26:WIDTH32
1110 LOCATE10,0:PRINT"INSTRUCCIONES"
1120 KEY1,"gosub1270"+CHR$(13)
1130 KEY2,"gosub1290"+CHR$(13)
1140 LOCATE2,3:PRINT"25 LINEAS POR PANTALLA
."
1150 LOCATE2,4:PRINT"USE LA LINEA INFERIOR
PARA":LOCATE2,5:PRINT"PULSAR F1 Y F2 SI HA
LLENADO"
1160 LOCATE2,6:PRINT"TODA LA PANTALLA."
1170 LOCATE5,8:PRINT"F1 MEMORIZA PANTALLA"
1180 LOCATE5,10:PRINT"F2 RESTAURA PANTALLA"
1190 LOCATE2,12:PRINT"EL PROGRAMA QUEDA LIB
RE."
1200 LOCATE2,13:PRINT"CONSTRUYA PANTALLAS M
EDIANTE "
1210 LOCATE2,14:PRINT"VPOKE O EN DIRECTO CO
N EL"
1220 LOCATE2,15:PRINT"TECLADO"
1230 PRINT:PRINT:PRINT"  PULSA UNA TECLA ";
:A$=INPUT$(1)
1240 VDP(4)=1:CLS:END
1250 CLS:GOTO 150
1260 SCREEN0:WIDTH37:KEYON:END
1270 POKE&HABD3,&H21:POKE&HABD6,&H11:POKE&H
ABDD,&H59
1280 A=USR1(0):RETURN
1290 POKE&HABD3,&H11:POKE&HABD6,&H21:POKE&H
ABDD,&H5C
1300 A=USR1(0):RETURN
```

Explicación del programa

- Líneas 10-160: preparan la pantalla y la RAM cargando las dos rutinas CM necesarias para la realización del programa. Se definen dos funciones USR: la primera para archivar en pantalla los caracteres o Sprites definidos por el usuario mediante el programa y la segunda para memorizar la posible pantalla a crear.
- Líneas 170-380: definen las cadenas del programa. Ponga especial cuidado para que la presentación en pantalla sea adecuada.
- Líneas 350-790: son el núcleo del programa. Observe el adecuado uso de las cadenas para presentar los datos introducidos en Binario, Decimal y Hexadecimal. También es de notar la dirección &HF3B1. Aquí cambiamos el número de líneas en pantalla de 24 a 3 según sea necesaria para mantener aislada la zona inferior de la pantalla de un posible error en la introducción de datos.

- Líneas 760-1090: presentan el menú de opciones y modifican las rutinas CM para la grabación de Sprites o caracteres. El algoritmo empleado es útil y económico ya que con unos simples "pokes" se ajustan las rutinsa CM a las necesidades requeridas. La línea 940 tiene las direcciones límite consideradas adecuadas por nosotros; pero puede cambiarlas.

El resto de las líneas definen las dos primeras teclas de función e indican las instrucciones para realizar una pantalla, quedando el programa libre. El programa puede ser relanzado en cualquier momento con RUN, sin ningún problema.

Capítulo 6

La potencia del MSX-2

Durante la afanosa confección del presente volumen, hemos tenido la grata sorpresa de tomar contacto con los nuevos MSX-2 y sus maravillosas posibilidades, especialmente gráficas. En seguida nos vimos envueltos, entre la curiosidad y el desafío, por la necesidad de comprobar todas sus capacidades.

No queremos, pues, terminar el presente libro dedicado al estándar MSX sin dedicar un capítulo en el que detallar, si no de un manera sistemática y exhaustiva que nos proporcionaría material más que suficiente para llenar otro volumen, sí, al menos, las principales características del MSX-2 y con ello describir los nuevos mandatos de que vienen dotados y que suponen un salto significativo tal en las posibilidades del MSX que no en vano demuestran que nos encontramos ante la segunda generación del estándar.

1. Funciones de inicialización

El MSX-2 BASIC ha sido dotado con unas funciones especiales que nos permiten cambiar ciertos parámetros iniciales tales como el ajuste de la zona de visualización, la introducción de una clave de acceso, volumen y tipo del Beep Sound, colores iniciales del anagrama de pantalla, etc. y que permanecerán en memoria aunque desconectemos la alimentación, gracias a una pequeña pila incorporada.

SET TITLE

Cuando el BASIC es inicializado, el ordenador saca en pantalla antes del mensaje Ok el siguiente anagrama:



VRAM: 128 KBYTES

Pues bien, podemos utilizar la instrucción:

SET TITLE "....." ,(color)

para que debajo del mensaje que indica la capacidad de la Videoram aparezca un rótulo de una extensión máxima de seis caracteres o también cambiar los colores del anagrama mediante un código de color.
Ejemplo:

SET TITLE"-RAMA-",2

La siguiente tabla muestra los colores permitidos para el segundo parámetro y las combinaciones generadas para tinta y fondo, por así decir.

MSX

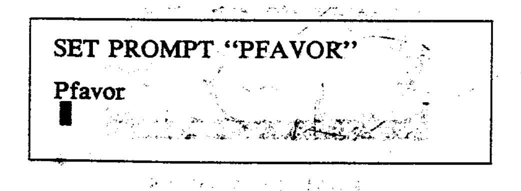
COLORES			
1	2	3	4
AZUL OSCURO	VERDE OSCURO	ROJO	AMARILLO OSCURO
NEGRO	AZUL OSCURO	MAGENTA	ROJO

SET PROMPT

Cuando el BASIC está disponible para aceptar un mandato en modo directo, el ordenador nos avisa con el ya conocido Ok. Pues bién, podemos cambiar dicho mensaje por cualquier otra palabra de una extensión máxima de seis caracteres.
Ejemplo:

SET PROMPT"Pfavor"

El mensaje quedará archivado y aparecerá siempre que no sea ejecutada otra instrucción del tipo Set Prompt, Set Password, Set Title ...



SET PASSWORD

Esta instrucción sirve para insertar una clave secreta de una longitud máxima de 256 caracteres que impedirá que el BASIC sea inicializado a no ser que introduzcamos correctamente la clave designada. Por tanto sólo aquél o aquellos que la conozcan estarán habilitados para usar el ordenador incluso aunque esté conectado un cartucho.
Ejemplo:

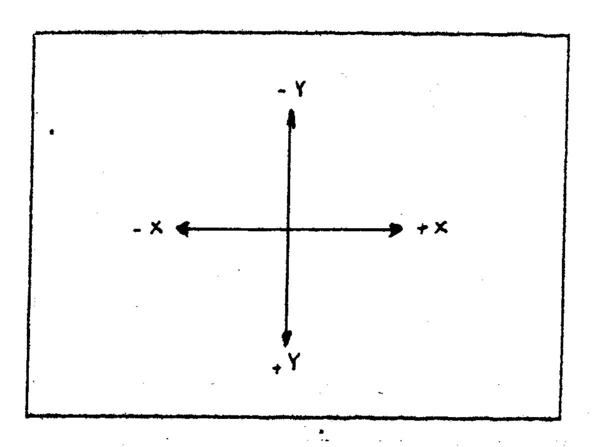
SET PASSWORD"RAMA,MADRID X54C"

Si por falta de memoria, el usuario olvida la clave podrá inicializar el sistema pulsando GRAPH STOP y RESET simultáneamente, o también GRAPH STOP al encender el ordenador y esperar así a que se inicialice el sistema.

SET ADJUST

Este mandato se utiliza para ajustar el área de pantalla en cualquiera de las cuatro direcciones desde un valor de −7 a +8 puntos de corrección. Los valores iniciales vienen dados a cero.
Ejemplo:

SET ADJUST (−3,4)



SET BEEP

Como sabemos, el "Beep Sound" aparece cada vez que ocurre un error en modo directo o en modo programa. Pues bien, en MSX-2 podemos elegir entre cuatro

modelos de sonido y cuatro puntos de volúmen para cada uno de ellos mediante la instrucción:

SET BEEP(modelo),(volumen)

SET BEEP (Modelo),(volumen)				
N.º	1	2	3	4
MODELO				
VOLUMEN	MÍNIMO	------	------	MÁXIMO

2. Los modos de pantalla

En MSX-2 BASIC disponemos de nueve modos de pantalla de los cuales los dos primeros son usados para texto y los demás para gráficos.

Los modos de texto (0 y 1) ya han sido explicados suficientemente en el capítulo tres. La única cosa importante que debemos de notar aquí es que en el modo Screen 0 tenemos la posibilidad de definir una anchura de hasta 80 caracteres en una línea y por lo tanto pasaremos a modo de 80 columnas cada vez que definamos un Width superior a 40.

En cuanto a los modos gráficos, podemos dividirlos en dos bloques:

- del 2-6 válidos para trabajar con una Videoram de 64K o de 128K indistintamente.
- 7 y 8 válidos sólo para trabajar con 128K de memoria de video.

De los modos 2 y 3, también hemos hablado ya, y aunque en realidad en todos los modos existen diferencias, no nos detendremos en ellas para estos dos modos hasta que hablemos del color.

El modo Screen 4 es igual que el modo Screen 2 pero admite la característica especial de poder trabajar con la función extendida para los Sprites. Lo veremos más adelante.

El modo Screen 5 trabaja con una matriz de 256×212 puntos de pantalla. Admite paginación (concepto que también veremos enseguida) y el modo extendido de Sprites, característica ésta que ya será común a todos los modos desde el 4 al 8. Este modo es el mejor para dibujar en pantalla debido a la rapidez que supone con respecto a los modos 2 y 4 a la hora de realizar los gráficos, y además soslaya el problema de la preferencia del último mandato, visto en el capítulo tercero.

A continuación ofrecemos un cuadro resumen de las características de todos los modos.

Modos de pantalla para MSX-2

N	MODOS	PANTALLA	COLOR	PÁGINAS	SPRITE
0	TEXTO	80×24 carac.	16/512	—	NO
1		32×24 carac.	16/512	—	SI
2	GRÁFICO 64/128K	256×192 puntos	16/512	—	SI
3		64×48 puntos	16/512	—	SI
4		256×192 puntos	16/512	—	EXTENDIDO
5		256×212 puntos	16/512	2/4	EXTENDIDO
6		512×212 puntos	16/512	2/4	EXTENDIDO
7	GRÁFICO Sólo 128K	512×212 puntos	16/512	2	EXTENDIDO
8		256×212 puntos	256	2	EXTENDIDO

Para acabar este punto vamos a ver un pequeño programa usando diferentes modos de pantalla para comprobar el efecto que se produce con el cambio de coordenadas de los modos 2, 5 y 6.

```
10 SCREEN 2
20 GOSUB 80
30 SCREEN 5
40 GOSUB 80
50 SCREEN 6
60 GOSUB 80
70 END
80 FOR I=1TO40STEP2:CIRCLE(100+I,90+I),10+
I:NEXT
90 RETURN
```

3. Los modos de pantalla y el color

Hasta ahora estábamos acostumbrados al uso limitado de 16 colores, para los cuales teníamos unos códigos preestablecidos. Esto sigue en pie, pero con la posibilidad de extenderlos de una forma increíble en una gama de tonalidades de hasta 512.

Los modos en los que podemos disponer de 16 colores (0 al 15) son: Screen 0, Screen 1, Screen 2, Screen 3, Screen 4, Screen 5 y Screen 7. Estos colores pueden usarse para crear 512 colores a la libre elección del usuario.

Como sería muy complicado trabajar con códigos determinados para cada uno de ellos, el método más sencillo es combinar, como si tuviéramos en realidad una paleta de pintor en la que mezclar diversos colores, la cantidad de rojo, verde y azul para producir el color deseado.

Por lo tanto, será su propia imaginación la que ponga límites a las posibilidades de color.

La característica antes descrita es denominada Paleta de Color.

Los tres colores de la Paleta (rojo, verde y azul) están dotados de ocho niveles de intensidad o brillo, numerados del cero al siete. Por tanto, combinando dichos niveles tendremos: 8*8*8=512 colores.

El color será negro cuando los niveles del rojo, verde y azul estén a cero y será blanco cuando los niveles estén al máximo valor, es decir a siete.

Color	Rojo	Verde	Azul
Negro	0	0	0
	1	1	1
Gris oscuro	2	2	2
	3	3	3
	4	4	4
Gris claro	5	5	5
	6	6	6
Blanco	7	7	7

Cuando la intensidad de uno de los colores es mayor que la de los otros dos colores, o la de éstos dos últimos es más cercana al valor cero, predominará el color de más alto nivel. Por ejemplo, 7,0,0 producirá un rojo puro y 2,0,0 determinará un rojo oscuro casi negro.

La instrucción usada para especificar el código de color y los diversos niveles para lograr un color especificado viene dada por los siguientes parámetros:

COLOR=(código,nivel de rojo,nivel de verde,nivel de azul)

teniendo en cuenta que los niveles no pueden exceder de 7.

Por ejemplo:

COLOR=(7,5,3,1)

asignará al código de color 7 un nivel 5 de rojo, 3 de verde y 1 de azul.

El modo Screen 6 y la Paleta de Color

Podemos usar la Paleta de Color en este modo pero de una manera un tanto limitada, pues sólo podemos disponer de cuatro códigos de color: del cero al tres.

Color	Código
Transparente	0
Negro	1
Verde	2
Verde claro	3

El modo Screen 8 y el color

En este modo no podemos usar la paleta de color, pero podemos disponer de 256 colores usando los códigos de color.

En Screen 8 disponemos de ocho niveles de intensidad para el rojo y el verde pero sólo cuatro para el azul, desde el 0 hasta el 3. Por lo tanto 8*8*4=256, tendremos 256 colores a nuestra disposición.

El código de color para cada uno de los disponibles hay que calcularlo con la siguiente fórmula:

Código de color=32*(nivel de verde)+4*(nivel del rojo)+(nivel del azul)

Por ejemplo:

Si el nivel del verde es 6, el del rojo 2 y el azul 3, el código de color sería: 32*6+4*2+3=203.

De los demás modos podemos concluir diciendo que el color en los modos Screen 2 y 4, podemos usar un máximo de dos colores para cada bloque específico de ocho puntos y que en los modos de pantalla del 5 al 8, los colores pueden ser libremente especificados para unidades de un pixel.

4. La paginación de pantallas

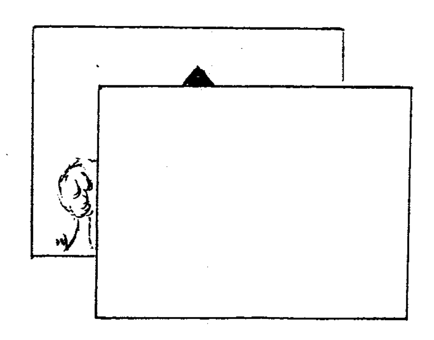
Desde el modo Screen 5 hasta el modo Screen 8, disponemos de una nueva posibilidad:

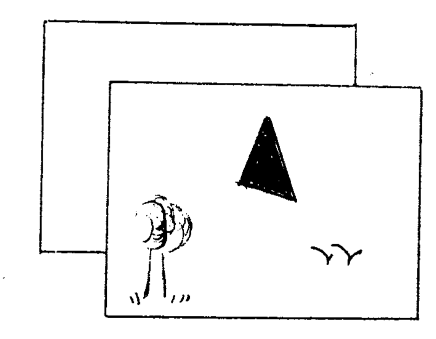
La paginación.

MODO	VRAM 64K	VRAM 128K
SCREEN 5	2 páginas	4 páginas
SCREEN 6	2 páginas	4 páginas
SCREEN 7	—	2 páginas
SCREEN 8	—	2 páginas

Como el nombre indica, el concepto de página es similar al de las hojas de un cuaderno que podemos pasar de una en una.

Como el cuadro indica, los ordenadores dotados de 64K de VRAM pueden disponer de dos páginas en los modos de pantalla 5 y 6. Cuando dibujamos un gráfico sólo una de las dos páginas es usada por el ordenador quedando la otra en blanco.





Cuando usamos dos páginas o más, dependiendo de la memoria de video, la página cero siempre será utilizada para visualizar los gráficos y también como zona de archivo de los mismos; a no ser que ordenemos otra cosa con el mandato:

SET PAGE

La página en la que realizamos los gráficos se denomina Página Activa y la página que vemos en pantalla se denomina Página Visible.

MSX-2 BASIC es capaz de seleccionar cada una de las páginas y así, dibujar en cualquiera de las páginas disponibles y mandarlas a pantalla cuando realmente queramos, haciendo que la página activa coincida con la página visible.

Usando esta característica, podemos obtener curiosos efectos entre los cuales destacamos la sensación de movimiento con la que podemos dotar a una página entera o a un dibujo solamente cambiando alternativamente de una página a otra.

El mandato SET PAGE tiene el siguiente formato:

SET PAGE(página visible),(página activa)

Ejemplo de ejecución:

SET PAGE 0,1

Cuando el mandato es ejecutado, 0 será la página visible y página 1 será la página donde se dibujen los gráficos.

5. Copia de gráficos

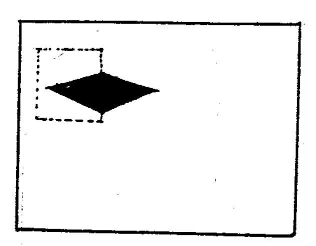
Los gráficos realizados en los modos 5, 6, 7 y 8 pueden ser copiados especificando un área determinada de la pantalla.

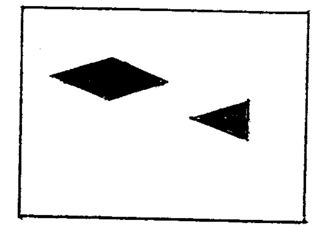
El bloque copiado puede ser transferido, según las siguientes combinaciones, a:

- Otra zona de la página
- Otra página
- La memoria interna
- Un archivo en disco

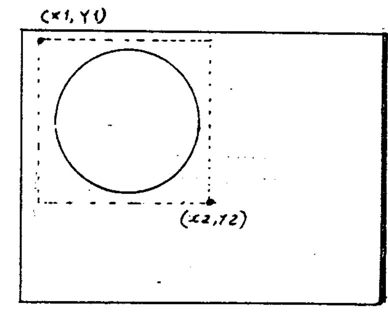
Veamos ahora cada una de las posibilidades.

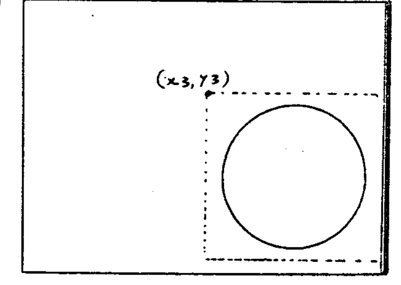
Copia en otra zona de la misma página





Copia en otra página





Como vemos en los gráficos anteriores, podemos copiar parte de un dibujo o todo él especificando las coordenadas inicial y final del área a copiar y las coordenadas iniciales del área de destino bien sea en la misma página o en otra de las disponibles.

El formato a que deberemos sujetarnos será el siguiente:

COPY (X1,Y1)-(X2,Y2),página origen TO (X3,Y3),página destino

Ejemplo de ejecución:

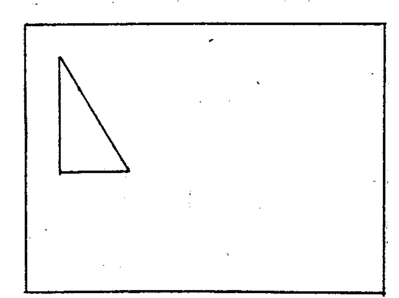
```
10 SCREEN 5
20 SET PAGE 0,0
30 FOR I=1TO40STEP2:CIRCLE(100+I,90+I),10+
I:NEXT
40 COPY(90,90)-(160,150),0 TO (50,50),1
50 SET PAGE 1
60 GOTO 60
```

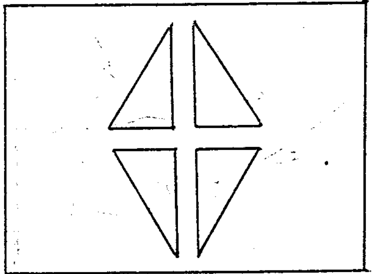
Si seguimos las líneas de programa notaremos como en la 20 seleccionamos la página 0 como página activa y también visible. Por tanto el dibujo se realizará ante nuestros ojos. En la línea 40 copiamos el bloque de pantalla indicado por las coordenadas de origen y lo situamos en la página 1 a partir de las coordenadas (50,50). Para ver si es cierto, en la línea 50 hacemos que la página visible sea la 1, precisamente aquélla a la que transferimos el bloque.

Copias de pantalla a memoria interna

En los casos anteriores, la copia de pantalla se realizaba de VRAM a VRAM, pero también podemos copiarlos en la RAM y al revés. Como vemos, no necesitamos crear rutinas como las que hemos podido ver en otra parte de este mismo libro. Aquí todo lo hace el sistema por nosotros.

Cuando pasamos datos de una copia de pantalla de la RAM a la VRAM, el sistema nos permite, además, girar la orientación del dibujo.





El proceso se realiza de la siguiente manera:

- Creamos una matriz de tipo numérico por medio de la instrucción DIM quee sea capaz de almacenar los datos de la zona de pantalla a copiar.
- Ejecutamos el mandato COPY bajo el siguiente formato:

COPY (X1,Y1)-(X2,Y2),página origen TO variable matriz.

Para dimensionar la matriz en su justa medida, podemos usar la siguiente fórmula:

INT ((((ABS(X1-X2)+1)*(ABS(X1-Y2)+1)*n.° de pixels+7)/8+4)/8)+1

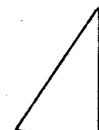
MODO	PIXELS
5	4
6	2
7	4
8	8

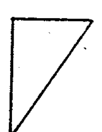
Para realizar la operación inversa, es decir, copiar de RAM a VRAM utilizaremos el siguiente formato:

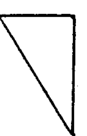
COPY varible,(orientación) TO (X3,Y3),página de destino

Disponemos de cuatro direcciones posibles numeradas del 0 al 3 en las que situar el inicio del dibujo:









Cuando el parámetro que indica la orientación se omite, el valor tomado será 0, por defecto.

Copias desde pantalla a Floppydisk

La instrucción COPY también puede ser utilizada para pasar una pantalla a Disco bajo el siguiente formato:

COPY(X1,Y1)–(X2,Y2),página origen TO "(nombre de unidad),nombre de archivo.(tipo)"

Al igual que en el punto anterior, cuando copiamos una zona de pantallas desde el disco podremos especificar uno de los cuatro tipos de orientación ya vistos.
El formato para la instrucción COPY será como sigue:

COPY "(nombre de unidad)nombre de archivo (.tipo)"(,orientación) TO (X3,Y3)(,página destino)

Copia de RAM a Floppydisk

Para copiar de la unidad de disco a una variable de matriz utilizaremos el siguiente formato:

COPY"(n.° unidad)n.° archivo(,tipo)" TO variable matriz

El proceso inverso sigue el siguiente formato:

COPY variable TO "(n.° unidad) n.° archivo(.tipo)"

Operaciones lógicas con la función COPY

Cuando realizamos copia de gráficos con la instrucción COPY podemos usar un último parámetro en el que incluiremos una de las siguientes operaciones lógicas:

PSET	PRESET	XOR	OR	AND
TPSET	TPRESET	TXOR	TOR	TAND

Las operaciones lógicas anteriores se realizan entre el código de color del motivo gráfico y el de la pantalla de destino. Si en el último parámetro de COPY no especificamos el tipo de operación lógica se adoptará por defecto la operación PSET.

6. Los Sprites

Otra de las gratas sorpresas que nos deparan los MSX-2 reside en el especial tratamiento de los Sprites.
Las diferencias esenciales con respecto a lo que ya sabemos sobre el tratamiento de sprites visto en el capítulo 4 se basan fundamentalmente en la capacidad extendida sobre el uso de Sprites y la posibilidad de crear sprites multicolores, para lo cual contamos además de la instrucción PUT SPRITE, con la valiosa ayuda de la función SPRITE COLOR.

En el cuadro adjunto trazamos una visión general de dichas diferencias.

FUNCIÓN	SCREEN 1-4	SCREEN 4-8
N.° de sprites por la línea horizontal	4 como máximo	8 como máximo
Colores	1 por sprite	8 para los de 8×8 o 16 para los de 16×16 como máximo por sprite
Cambio de color	Utilizando la función PUT SPRITE	Utilizando PUT SPRITE o COLOR SPRITE

El color de un sprite puede ser especificado con la función PUT SPRITE; pero desde los modos 4 hasta el 8 el color de un sprite puede ser cambiado una vez definido con la función COLOR SPRITE.

COLOR SPRITE(n.° de plano),color

Pero lo más importante es que podemos asignar un color para cada una de las ocho líneas de ocho puntos que componen un Sprite de 8×8· o para cada una de las 16 líneas de 16 puntos que componen un Sprite de 16×16.
Para esta facilidad usaremos:

COLOR SPRITE$(n.° de plano)=CHR$(código color)

Si queremos solamente cambiar el color de las tres primeras líneas para el Sprite de plano 0, ejecutaríamos el mandato de la siguiente manera:

COLOR SPRITE$=(0)=CHR$(X)+CHR$(Y)+CHR$(Z)

Las líneas no especificadas no cambiarán en absoluto.

7. Archivos y dispositivos archivadores

MSX-2 cuenta con los siguientes dispositivos archivadores

DISPOSITIVO	NOMBRE
Cassette	CAS:
Pantalla modo texto	CRT:
Pantalla modo gráfico	GRP:
Impresora	LPT:
Floppydisk A, B ... H	A:, B: ... H:
Memory disk	MEM:

La función Memory Disk

En MSX-2, además del área de programa, existe una zona de memoria que puede ser utilizada como archivo temporal y por tanto, podremos guardar en ella desde un bloque de datos a un programa entero. Esta zona tiene la particularidad de ser usada por el ordenador como si en realidad fuera un Floppydisk, pero con la ventaja de un acceso más rápido.

El área a la que nos referimos se denomina Memory Disk. Para usarla, previamente deberemos inicializarla con la instrucción:

CALL MEMINI (tamaño)

El tamaño de memoria puede ser fijado desde 1023 bytes a 32767, es decir podemos seleccionar una extensión de hasta 32K. Por defecto, el ordenador asignará la máxima extensión.

Cuando hacemos esta llamada, nos aparecerá en pantalla el tamaño especificado para Memory Disk.

Una vez realizada la operación anterior, podremos guardar, en el área reservada un programa que tengamos cargado en el área de programa, y recuperarlo según los formatos habituales.

Ejemplos:

- SAVE"MEM:RAMA.BAS"
- LOAD"MEM:RAMA.BAS"
- MERGE"MEM:RAMA.BAS"

Manejo de archivos

Deberíamos aquí explicar los mandatos disponibles del DISK BASIC, pero los damos por supuestos. No obstante, hacemos referencia a los directamente relacionados con la función Memory Disk y que serían los siguientes:

- CALL MFILES: Para sacar un directorio de los archivos presentes y al mismo tiempo los bytes disponibles
- CALL MKILL: Para borrar un archivo presente en Memory Disk
- CALL MNAME: Para cambiar de nombre un archivo presente en Memory Disk.

Apéndice 1

Clasificación de los códigos de operación por orden numérico

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
00	NOP	2D	DEC L	5A	LD E,D
018405	LD BC,NN	2E20	LD L,N	5B	LD E,E
02	LD (BC),A	2F	CPL	5C	LD E,H
03	INC BC	302E	JR NC,DIS	5D	LD E,L
04	INC B	318405	LD SP,NN	5E	LD E,(HL)
05	DEC B	328405	LD (NN),A	5F	LD E,A
0620	LD B,N	33	INC SP	60	LD H,B
07	RLCA	34	INC (HL)	61	LD H,C
08	EX AF,AF'	35	DEC (HL)	62	LD H,A
09	ADD HL,BC	3620	LD (HL),N	63	LD H,E
0A	LD A,(BC)	37	SCF	64	LD H,H
0B	DEC BC	382E	JR C,DIS	65	LD H,L
0C	INC C	39	ADD HL,SP	66	LD H,(HL)
0D	DEC C	3A8405	LD A,(NN)	67	LD H,A
0E20	LD C,N	3B	DEC SP	68	LD L,B
0F	RRCA	3C	INC A	69	LD L,C
102E	DJNZ DIS	3D	DEC A	6A	LD L,D
118405	LD DE,NN	3E20	LD A,N	6B	LD L,E
12	LD (DE),A	3F	CCF	6C	LD L,H
13	INC DE	40	LD B,B	6D	LD L,L
14	INC D	41	LD B,C	6E	LD L,(HL)
15	DEC D	42	LD B,D	6F	LD L,A
1620	LD D,N	43	LD B,E	70	LD (HL),B
17	RLA	44	LD B,H,NN	71	LD (HL),C
182E	JR DIS	45	LD B,L	72	LD (HL),D
19	ADD HL,DE	46	LD B,(HL)	73	LD (HL),E
1A	LD A,(DE)	47	LD B,A	74	LD (HL),H
1B	DEC DE	48	LD C,B	75	LD (HL),L
1C	INC E	49	LD C,C	76	HALT
1D	DEC E	4A	LD C,D	77	LD (HL),A
1E20	LD E,N	4B	LD C,E	78	LD A,B
1F	RRA	4C	LD C,H	79	LD A,C
202E	JR NZ,DIS	4D	LD C,L	7A	LD A,D
218405	LD HL,NN	4E	LD C,(HL)	7B	LD A,E
228405	LD (NN),HL	4F	LD C,A	7C	LD A,H
23	INC HL	50	LD D,B	7D	LD A,L
24	INC H	51	LD D,C	7E	LD A,(HL)
25	DEC H	52	LD D,D	7F	LD A,A
2620	LD H,N	53	LD D,E	80	ADD A,B
27	DAA	54	LD D,H	81	ADD A,C
282E	JR Z,DIS	55	LD D,L	82	ADD A,D
29	ADD HL,HL	56	LD D,(HL)	83	ADD A,E
2A8405	LD (HL),(NN)	57	LD D,A	84	ADD A,H
2B	DEC HL	58	LD E,B	85	ADD A,L
2C	INC L	59	LD E,C	86	ADD A,(HL)

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
87	ADD A,A	CA8405	JP Z,NN	CB11	RL C
88	ADC A,B	CC8405	CALL Z,NN	CB12	RL D
89	ADC A,C	CD8405	CALL NN	CB13	RL E
8A	ADC A,D	CE20	ADC A,N	CB14	RL H
8B	ADC A,E	CF	RST 8	CB15	RL L
8C	ADC A,H	D0	RET NC	CB16	RL (HL)
8D	ADC A,L	D1	POP DE	CB17	RL A
8E	ADC A,(HL)	D28405	JP NC,NN	CB18	RR B
8F	ADC A,A	D320	O0T (N),A	CB19	RR C
90	SUB B	D48405	CALL NC,NN	CB1A	RR L
91	SUB C	D5	PUSH DE	CB1B	RR E
92	SUB D	D620	SUB N	CB1C	RR H
93	SUB E	D7	RST 10H	CB1D	RR L
94	SUB H	D8	RET C	CB1E	RR (HL)
95	SUB L	D9	EXX	CB1F	RR A
96	SUB (HL)	DA8405	JP C,NN	CB20	SLA B
97	SUB A	DB20	IN A, (N)	CB21	SLA C
98	SBC A,B	DC8405	CALL C,N	CB22	SLA D
99	SBC A,C	DE20	SBC A,N	CB23	SLA E
9A	SBC A,D	DF	RST 18H	CB24	SLA H
9B	SBC A,E	E0	RET PO	CB25	SLA L
9C	SBC A,H	E1	POP HL	CB26	SLA (HL)
9D	SBC A,L	E28405	JP PO,NN	CB27	SLA A
9E	SBC A,(HL)	E3	EX (SP),HL	CB28	SRA B
9F	SBC A,A	E48405	CALL PO,NN	CB29	SRA C
A0	AND B	E5	PUSH HL	CB2A	SRA D
A1	AND C	E620	AND N	CB2B	SRA E
A2	AND D	E7	RST 20H	CB2C	SRA H
A3	AND E	E8	RET PE	CB2D	SRA L
A4	AND H	E9	JP (HL)	CB2E	SRA (HL)
A5	AND L	EA8405	JE PE NN	CB2F	SRA A
A6	AND (HL)	EB	EX DE,HL	CB38	SRL B
A7	AND A	EC8405	CALL PE,NN	CB39	SRL C
A8	XOR B	EE20	XOR N	CB3A	SRL D
A9	XOR C	EF	RST 28H	CB3B	SRL E
AA	XOR D	F0	RET P	CB3C	SRL H
AB	XOR E	F1	POP AF	CB3D	SRL L
AC	XOR H	F28405	JP P,NN	CB3E	SRL (HL)
AD	XOR L	F3	DI	CB3F	SRL A
AE	XOR (HL)	F48405	CALL P,NN	CB40	BIT 0,B
AF	XOR A	F5	PUSH AF	CB41	BIT 0,C
B0	OR B	F620	OR N	CB42	BIT 0,D
B1	OR C	F7	RST 30H	CB43	BIT 0,E
B2	OR D	F8	RET M	CB44	BIT 0,H
B3	OR E	F9	LD SP,HL	CB45	BIT 0,L
B4	OR H	FA8405	JP M,NN	CB46	BIT 0,(HL)
B5	OR L	FB	EI	CB47	BIT 0,A
B6	OR (HL)	FC8405	CALL M,NN	CB48	BIT 1,B
B7	OR A	FE20	CP N	CB49	BIT 1,C
B8	CP B	FF	RST 38H	CB4A	BIT 1,D
B9	CP C	CB00	RLC B	CB4B	BIT 1,E
BA	CP D	CB01	RLC C	CB4C	BIT 1,H
BB	CP E	CB02	RLC D	CB4D	BIT 1,L
BC	CP H	CB03	RLC E	CB4E	BIT 1,(HL)
BD	CP L	CB04	RLC H	CB4F	BIT 1,A
BE	CP (HL)	CB05	RLC L	CB50	BIT 2,B
BF	CP A	CB06	RLC (HL)	CB51	BIT 2,C
C0	RET NZ	CB07	RLC A	CB52	BIT 2,D
C1	POP BC	CB08	RRC B	CB53	BIT 2,E
C28405	JP NZ,NN	CB09	RRC C	CB54	BIT 2,H
C38405	JP NN	CB0A	RRC D	CB55	BIT 2,L
C48405	CALL NZ,NN	CB0B	RRC E	CB56	BIT 2,(HL)
C5	PUSH BC	CB0C	RRC H	CB57	BIT 2,A
C620	ADD A,N	CB0D	RRC L	CB58	BIT 3,B
C7	RST 0	CB0E	RRC (HL)	CB59	BIT 3,C
C8	RET Z	CB0F	RRC A	CB5A	BIT 3,D
C9	RET	CB10	RL B		

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
CB5B	BIT 3,E	CB9E	RES 3,(HL)	CBE1	SET 4,C
CB5C	BIT 3,H	CB9F	RES 3,A	CBE2	SET 4,D
CB5D	BIT 3,L	CBA0	RES 4,B	CBE3	SET 4,E
CB5E	BIT 3,(HL)	CBA1	RES 4,C	CBE4	SET 4,H
CB5F	BIT 3,A	CBA2	RES 4,D	CBE5	SET 4,L
CB60	BIT 4,B	CBA3	RES 4,E	CBE6	SET 4,(HL)
CB61	BIT 4,C	CBA4	RES 4,H	CBE7	SET 4,A
CB62	BIT 4,D	CBA5	RES 4,L	CBE8	SET 5,B
CB63	BIT 4,E	CBA6	RES 4,(HL)	CBE9	SET 5,C
CB64	BIT 4,H	CBA7	RES 4,A	CBEA	SET 5,D
CB65	BIT 4,L	CBA8	RES 5,B	CBEB	SET 5,E
CB66	BIT 4,(HL)	CBA9	RES 5,C	CBEC	SET 5,H
CB67	BIT 4,A	CBAA	RES 5,D	CBED	SET 5,L
CB68	BIT 5,B	CBAB	RES 5,E	CBEE	SET 5,(HL)
CB69	BIT 5,C	CBAC	RES 5,H	CBEF	SET 5,A
CB6A	BIT 5,D	CBAD	RES 5,L	CBF0	SET 6,B
CB6B	BIT 5,E	CBAE	RES 5,(HL)	CBF1	SET 6,C
CB6C	BIT 5,H	CBAF	RES 5,A	CBF2	SET 6,D
CB6D	BIT 5,L	CBB0	RES 6,B	CBF3	SET 6,E
CB6E	BIT 5,(HL)	CBB1	RES 6,C	CBF4	SET 6,H
CB6F	BIT 5,A	CBB2	RES 6,D	CBF5	SET 6,L
CB70	BIT 6,B	CBB3	RES 6,E	CBF6	SET 6,(HL)
CB71	BIT 6,C	CBB4	RES 6,H	CBF7	SET 6,A
CB72	BIT 6,D	CBB5	RES 6,L	CBF8	SET 7,B
CB73	BIT 6,E	CBB6	RES 6,(HL)	CBF9	SET 7,C
CB74	BIT 6,H	CBB7	RES 6,A	CBFA	SET 7,D
CB75	BIT 6,L	CBB8	RES 7,B	CBFB	SET 7,E
CB76	BIT 6,(HL)	CBB9	RES 7,C	CBFC	SET 7,H
CB77	BIT 6,A	CBBA	RES 7,D	CBFD	SET 7,L
CB78	BIT 7,B	CBBB	RES 7,E	CBFE	SET 7,(HL)
CB79	BIT 7,C	CBBC	RES 7,H	CBFF	SET 7,A
CB7A	BIT 7,D	CBBD	RES 7,L	DD09	ADD IX,BC
CB7B	BIT 7,E	CBBE	RES 7,(HL)	DD19	ADD IX,DE
CB7C	BIT 7,H	CBBF	RES 7,A	DD218405	LD IX,NN
CB7D	BIT 7,L	CBC0	SET 0,B	DD228405	LD (NN),IX
CB7E	BIT 7,(HL)	CBC1	SET 0,C	DD23	INC IX
CB7F	BIT 7,A	CBC2	SET 0,D	DD29	ADD IX,IX
CB80	RES 0,B	CBC3	SET 0,E	DD2A8405	LD IX,(NN)
CB81	RES 0,C	CBC4	SET 0,H	DD2B	DEC IX
CB82	RES 0,D	CBC5	SET 0,L	DD3405	INC (IX+d)
CB83	RES 0,E	CBC6	SET 0,(HL)	DD3505	DEC (IX+d)
CB84	RES 0,H	CBC7	SET 0,A	DD360520	LD (IX+d),N
CB85	RES 0,L	CBC8	SET 1,B	DD39	ADD IX,SP
CB86	RES 0,(HL)	CBC9	SET 1,C	DD4605	LD B,(IX+d)
CB87	RES 0,A	CBCA	SET 1,D	DD4E05	LD C,(IX+d)
CB88	RES 1,B	CBCB	SET 1,E	DD5605	LD D, (IX+d)
CB89	RES 1,C	CBCC	SET 1,H	DD5E05	LD E,(IX+d)
CB8A	RES 1,D	CBCD	SET 1,L	DD6605	LD H,(IX+d)
CB8B	RES 1,E	CBCE	SET 1,(HL)	DD6E05	LD L,(IX+d)
CB8C	RES 1,H	CBCF	SET 1,A	DD7005	LD (IX+d),B
CB8D	RES 1,L	CBD0	SET 2,B	DD7105	LD (IX+d),C
CB8E	RES 1,(HL)	CBD1	SET 2,C	DD7205	LD (IX+d),D
CB8F	RES 1,A	CBD2	SET 2,D	DD7305	LD (IX+d),E
CB90	RES 2,B	CBD3	SET 2,E	DD7405	LD (IX+d),H
CB91	RES 2,C	CBD4	SET 2,H	DD7505	LD (IX+d),L
CB92	RES 2,D	CBD5	SET 2,L	DD7705	LD (IX+d),A
CB93	RES 2,E	CBD6	SET 2,(HL)	DD7E05	LD A,(IX+d),A
CB94	RES 2,H	CBD7	SET 2,A	DD8605	ADD A,(IX+d)
CB95	RES 2,L	CBD8	SET 3,B	DD8E05	ADC A,(IX+d)
CB96	RES 2,(HL)	CBD9	SET 3,C	DD9605	SUB (IX+d)
CB97	RES 2,A	CBDA	SET 3,D	DD9E05	SBC A,(IX+d)
CB98	RES 3,B	CBDB	SET 3,E	DDA605	AND (IX+d)
CB99	RES 3,C	DBDC	SET 3,H	DDAE05	XOR (IX+d)
CB9A	RES 3,D	CBDD	SET 3,L	DDB605	OR (IX+d)
CB9B	RES 3,E	CBDE	SET 3,(HL)	DDBE05	CP (IX+d)
CB9C	RES 3,H	CBDF	SET 3,A	DDE1	POP IX
CB9D	RES 3,L	CBE0	SET 4,B	DDE3	EX (SP),IX

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
DDE5	PUSH IX	ED738405	LD (NN),SP	FDCB0546	BIT 0,(IY+d)
DDE9	JP (IX)	ED78	IN A,(C)	FDCB054E	BIT 1,(IY+d)
DDF9	LD SP,IX	ED79	OUT (C),A	FDCB0556	BIT 2,(IY+d)
DDCB0506	RLC (IX+d)	ED7A	ADC HL,SP	FDCB055E	BIT 3,(IY+d)
DDCB050E	RRC (IX+d)	ED7B8405	LD SP,(NN)	FDCB0566	BIT 4,(IY+d)
DDCB0516	RL (IX+d)	EDA0	LDI	FDCB056E	BIT 5,(IY+d)
DDCB051E	RR (IX+d)	EDA1	CPI	FDCB0576	BIT 6,(IY+d)
DDCB0526	SLA (IX+d)	EDA2	INI	FDCB057E	BIT 7,(IY+d)
DDCB052E	SRA (IX+d)	EDA3	OUTI	FDCB0586	RES 0,(IY+d)
DDCB053E	SRL (IX+d)	EDA8	LDD	FDCB058E	RES 1,(IY+d)
DDCB0546	BIT 0,(IX+d)	EDA9	CPD	FDCB0596	RES 2,(IY+d)
DDCB054E	BIT 1,(IX+d)	EDAA	IND	FDCB059E	RES 3,(IY+d)
DDCB0556	BIT 2,(IX+d)	EDAB	OUTD	FDCB05A6	RES 4,(IY+d)
DDCB055E	BIT 3,(IX+d)	EDB0	LDIR	FDCB05AE	RES 5,(IY+d)
DDCB0566	BIT 4,(IX+d)	EDB1	CPIR	FDCB05B6	RES 6,(IY+d)
DDCB056E	BIT 5,(IX+d)	EDB2	INIR	FDCB05BE	RES 7,(IY+d)
DDCB0576	BIT 6,(IX+d)	EDB3	OTIR	FDCB05C6	SET 0,(IY+d)
DDCB057E	BIT 7,(IX+d)	EDB8	LDDR	FDCB05CE	SET 1,(IY+d)
DDCB0586	RES 0,(IX+d)	EDB9	CPDR	FDCB05D6	SET 2,(IY+d)
DDCB058E	RES 1,(IX+d)	EDBA	INDR	FDCB05DE	SET 3,(IY+d)
DDCB0596	RES 2,(IX+d)	EDBB	OTDR	FDCB05E6	SET 4,(IY+d)
DDCB059E	RES 3,(IX+d)	FD09	ADD IY,BC	FDCB05EE	SET 5,(IY+d)
DDCB05A6	RES 4,(IX+d)	FD19	ADD IY,DE	FDCB05F6	SET 6,(IY+d)
DDCB05AE	RES 5,(IX+d)	FD218405	LD IY,NN	FDCB05FE	SET 7,(IY+d)
DDCB05B6	RES 6,(IX+d)	FD228405	LD (NN),IY		
DDCB05BE	RES 7,(IX+d)	FD23	INC IY		
DDCB05C6	SET 0,(IX+d)	FD29	ADD IY,IY		
DDCB05CE	SER 1,(IX+d)	FD2A8405	LD IY,(NN)		
DDCB05D6	SER 2,(IX+d)	FD2B	DEC IY		
DDCB05DE	SET 3,(IX+d)	FD3405	INC (IY+d)		
DDCB05E6	SET 4,(IX+d)	FD3505	DEC(IY+d)		
DDCB05EE	SET 5,(IX+d)	FD360520	LD (IY+d),N		
DDCB05F6	SET 6,(IX+d)	FD39	ADD IY,SP		
DDCB05FE	SET 7,(IX+d)	FD4605	LD B,(IY+d)		
ED40	IN B,(C)	FD4E05	LD C,(IY+d)		
ED41	OUT (C),B	FD5605	LD D,(IY+d)		
ED42	SBC HL,BC	FD5E05	LD E,(IY+d)		
ED438405	LD (NN),BC	FD6605	LD H,(IY+d)		
ED44	NEG	FD6E05	LD L,(IY+d)		
ED45	RETN	FD7005	LD (IY+d),B		
ED46	IM 0	FD7105	LD (IY+d),C		
ED47	LD I,A	FD7205	LD (IY+d),D		
ED48	IN C,(C)	FD7305	LD (IY+d),E		
ED49	OUT (C),C	FD7405	LD (IY+d),H		
ED4A	ADC HL,BC	FD7505	LD (IY+d),L		
ED4B8405	LD BC,(NN)	FD7705	LD (IY+d),A		
ED4D	RETI	FD7E05	LD A,(IY+d)		
ED50	IN D,(C)	FD8605	ADD A,(IY+d)		
ED51	OUT (C),D	FD8E05	ADC A,(IY+d)		
ED52	SBC HL,DE	FD9605	SUB (IY+d)		
ED538405	LD (NN),DE	FD9E05	SBC A,(IY+d)		
ED56	IM 1	FDA605	AND (IY+d)		
ED57	LD A,I	FDAE05	XOR (IY+d)		
ED58	IN E,(C)	FDB605	OR (IY+d)		
ED59	OUT (C),E	FDBE05	CP (IY+d)		
ED5A	ADC HL,DE	FDE1	POP IY		
ED5B8405	LD DE,(NN)	FDE3	EX (SP),IY		
ED5E	IM 2	FDE5	PUSH IY		
ED60	IM H,(C)	FDE9	JP (IY)		
ED61	OUT (C),H	FDF9	LD SP,IY		
ED62	SBC HL,HL	FDCB0506	RLC (IY+d)		
ED67	RRD	FDCB050E	RRC (IY+d)		
ED68	IN L,(C)	FDCB0516	RL (IY+d)		
ED69	OUT (C),L	FDCB051E	RR (IY+d)		
ED6A	ADC HL,HL	FDCB0526	SLA (IY+d)		
ED6F	RLD	FDCB052E	SRA (IY+d)		
ED72	SBC HL,SP	FDCB053E	SRL (IY+d)		

Apéndice 2

Clasificación de los códigos de operación por orden alfabético

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
8E	ADC A,(HL)	A3	AND E	DDCB0566	BIT 4,(IX+d)
DD8E05	ADC A,(IX+d)	A4	AND H	FDCB0566	BIT 4,(IY+d)
FD8E05	ADC A,(IY+d)	A5	AND L	CB67	BIT 4,A
8F	ADC A,A	E620	AND N	CB60	BIT 4,B
88	ADC A,B	CB46	BIT 0,(HL)	CB61	BIT 4,C
89	ADC A,C	DDCB0546	BIT 0,(IX+d)	CB62	BIT 4,D
8A	ADC A,D	FDCB0546	BIT 0,(IY+d)	CB63	BIT 4,E
8B	ADC A,E	CB47	BIT 0,A	CB64	BIT 4,H
8C	ADC A,H	CB40	BIT 0,B	CB65	BIT 4,L
8D	ADC A,L	CB41	BIT 0,C	CB6E	BIT 5,(HL)
CE20	ADC A,N	CB42	BIT 0,D	DDCB056E	BIT 5,(IX+d)
ED4A	ADC HL,BC	CB43	BIT 0,E	FDCB056E	BIT 5,(IY+d)
ED5A	ADC HL,DE	CB44	BIT 0,H	CB6F	BIT 5,A
ED6A	ADC HL,HL	CB45	BIT 0,L	CB68	BIT 5,B
ED7A	ADC HL,SP	CB4E	BIT 1,(HL)	CB69	BIT 5,C
86	ADD A,(HL)	DDCB054E	BIT 1,(IX+d)	CB6A	BIT 5,D
DD8605	ADD A,(IX+d)	FDCB054E	BIT 1,(IY+d)	CB6B	BIT 5,E
FD8605	ADD A,(IY+d)	CB4F	BIT 1,A	CB6C	BIT 5,H
87	ADD A,A	BC48	BIT 1,B	CB6D	BIT 5,L
80	ADD A,B	CB49	BIT 1,C	CB76	BIT 6,(HL)
81	ADD A,C	CB4A	BIT 1,D	DDCB0576	BIT 6,(IX+d)
82	ADD A,D	CB4B	BIT 1,E	FDCB0576	BIT 6,(IY+d)
83	ADD A,E	CB4C	BIT 1,H	CB77	BIT 6,A
84	ADD A,H	CB4D	BIT 1,L	CB70	BIT 6,B
85	ADD A,L	CB56	BIT 2,(HL)	CB71	BIT 6,C
C620	ADD A,N	DDCB0556	BIT 2,(IX+d)	CB72	BIT 6,D
09	ADD HL,BC	FDCB0556	BIT 2,(IY+d)	CB73	BIT 6,E
19	ADD HL,DE	CB57	BIT 2,A	CB74	BIT 6,H
29	ADD HL,HL	CB50	BIT 2,B	CB75	BIT 6,L
39	ADD HL,SP	CB51	BIT 2,C	CB7E	BIT 7,(HL)
DD09	ADD IX,BC	CB52	BIT 2,D	DDCB057E	BIT 7,(IX+d)
DD19	ADD IX,DE	CB53	BIT 2,E	FDCB057E	BIT 7,(IY+d)
DD29	ADD IX,IX	CB54	BIT 2,H	CB7F	BIT 7,A
DD39	ADD IX,SP	CB55	BIT 2,L	CB78	BIT 7,B
FD09	ADD IY,BC	CB5E	BIT 3,(HL)	CB79	BIT 7,C
FD19	ADD IY,DE	DDCB055E	BIT 3,(IX+d)	CB7A	BIT 7,D
FD29	ADD IY,IY	FDCB055E	BIT 3,(IY+d)	CB7B	BIT 7,E
FD39	ADD IY,SP	CB5F	BIT 3,A	CB7C	BIT 7,H
A6	AND (HL)	CB58	BIT 3,B	CB7D	BIT 7,L
DDA605	AND (IX+d)	CB59	BIT 3,C	DC8405	CALL C,NN
FDA605	AND (IY+d)	CB5A	BIT 3,D	FC8405	CALL M,NN
A7	AND A	CB5B	BIT 3,E	D48405	CALL NC,NN
A0	AND B	CB5C	BIT 3,H	CD8405	CALL NN
A1	AND C	CB5D	BIT 3,L	C48405	CALL NZ,NN
A2	AND D	CB66	BIT 4,(HL)	F48405	CALL P,NN

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
EC8405	CALL PE,NN	1C	INC E	7F	LD A,A
E48405	CALL PO,NN	24	INC H	78	LD A,B
CC8405	CALL Z,NN	23	INC HL	79	LD A,C
3F	CCF	DD23	INC IX	7A	LD A,D
BE	CP (HL)	FD23	INC IY	7B	LD A,E
DDBE05	CP (IX+d)	2C	INC L	7C	LD A,H
FDBE05	CP (IY+d)	33	INC SP	ED57	LD A,I
BF	CP A	EDAA	IND	7D	LD A,L
B8	CP B	EDBA	INDR	3E20	LD A,N
B9	CP C	EDA2	INI	46	LD B,(HL)
BA	CP D	EDB2	INIR	DD4605	LD B,(IX+d)
BB	CP E	E9	JP (HL)	FD4605	LD B,(IY+d)
BC	CP H	DDE9	JP (IX)	47	LD B,A
BD	CP L	FDE9	JP (IY)	40	LD B,B
FE20	CP N	DA8405	JP C,NN	41	LD B,C
EDA9	CPD	FA8405	JP M,NN	42	LD B,D
EDB9	CPDR	D28405	JP NC,NN	43	LD B,E
EDA1	CPI	C38405	JP NN	44	LD B,H,NN
EDB1	CPIR	C28405	JP NZ,NN	45	LD B,L
2F	CPL	F28405	JP P,NN	0620	LD B,N
27	DAA	EA8405	JP PE,NN	ED4B8405	LD BC,(NN)
35	DEC (HL)	E28405	JP PO,NN	018405	LD BC,NN
DD3505	DEC (IX+d)	CA8405	JP Z,NN	4E	LD C,(HL)
FD3505	DEC (IY+d)	382E	JR C,DIS	DD4E05	LD C,(IX+d)
3D	DEC A	182E	JR DIS	FD4E05	LD C,(IY+d)
05	DEC B	302E	JR NC,DIS	4F	LD C,A
0B	DEC BC	202E	JR NZ,DIS	48	LD C,B
0D	DEC C	282E	JR Z,DIS	49	LD C,C
15	DEC D	02	LD (BC),A	4A	LD C,D
1B	DEC DE	12	LD (DE),A	4B	LD C,E
1D	DEC E	77	LD (HL),A	4C	LD C,H
25	DEC H	70	LD (HL),B	4D	LD C,L
2B	DEC HL	71	LD (HL),C	0E20	LD C,N
DD2B	DEC IX	72	LD (HL),D	56	LD D,(HL)
FD2B	DEC IY	73	LD (HL),E	DD5605	LD D,(IX+d)
2D	DEC L	74	LD (HL),H	FD5605	LD D,(IY+d)
3B	DEC SP	75	LD (HL),L	57	LD D,A
F3	DI	3620	LD (HL),N	50	LD D,B
102E	DJNZ DIS	DD7705	LD (IX+d),A	51	LD D,C
FB	EI	DD7005	LD (IX+d),B	52	LD D,D
E3	EX (SP),HL	DD7105	LD (IX+d),C	53	LD D,E
DDE3	EX (SP),IX	DD7205	LD (IX+d),D	54	LD D,H
FDE3	EX (SP),IY	DD7305	LD (IX+d),E	55	LD D,L
08	EX AF,AF'	DD7405	LD (IX+d),H	1620	LD D,N
EB	EX DE,HL	DD7505	LD (IX+d),L	ED5B8405	LD DE,(NN)
D9	EXX	DD360520	LD (IX+d),N	118405	LD DE,NN
76	HALT	FD7705	LD (IY+d),A	5E	LD E,(HL)
ED46	IM 0	FD7005	LD (IY+d),B	DD5E05	LD E,(IX+d)
ED56	IM 1	FD7105	LD (IY+d),C	FD5E05	LD E,(IY+d)
ED5E	IM 2	FD7205	LD (IY+d),D	5F	LD E,A
ED78	IN A,(C)	FD7305	LD (IY+d),E	58	LD E,B
DB20	IN A,(N)	FD7405	LD (IY+d),H	59	LD E,C
ED40	IN B,(C)	FD7505	LD (IY+d),L	5A	LD E,D
ED48	IN C,(C)	FD360520	LD (IY+d),N	5B	LD E,E
ED50	IN D,(C)	328405	LD (NN),A	5C	LD E,H
ED58	IN E,(C)	ED438405	LD (NN),BC	5D	LD E,L
ED60	IN H,(C)	ED538405	LD (NN),DE	1E20	LD E,N
ED68	IN L,(C)	228405	LD (NN),HL	66	LD H,(HL)
34	INC (HL)	DD228405	LD (NN),IX	DD6605	LD H,(IX+d)
DD3405	INC (IX+d)	FD228405	LD (NN),IY	FD6606	LD H,(IY+d)
FD3405	INC (IY+d)	ED738405	LD (NN),SP	67	LD H,A
3C	INC A	0A	LD A,(BC)	60	LD H,B
04	INC B	1A	LD A,(DE)	61	LD H,C
03	INC BC	7E	LD A,(HL)	62	LD H,D
0C	INC C	DD7E05	LD A,(IX+d)	63	LD H,E
14	INC D	FD7E05	LD A,(IY+d)	64	LD H,H
13	INC DE	3A8405	LD A,(NN)	65	LD H,L

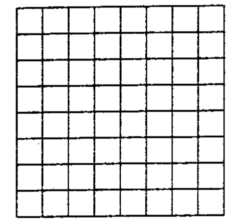
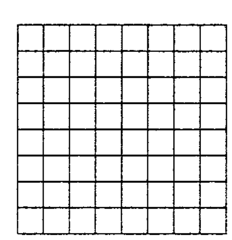
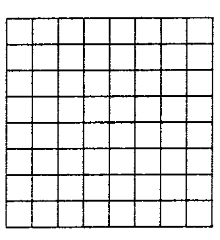
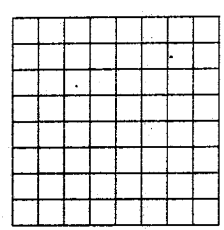
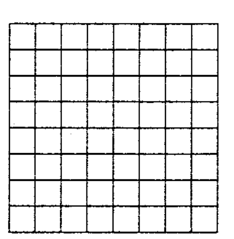
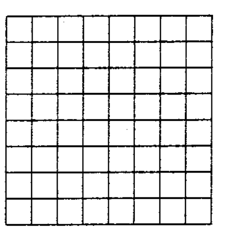
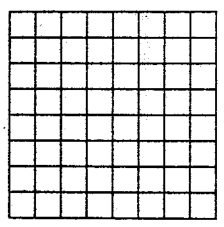
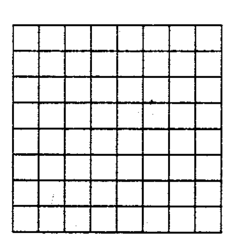
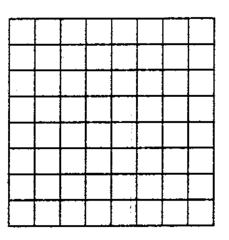
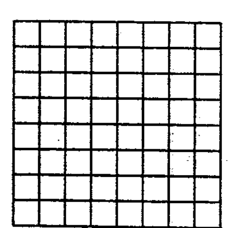
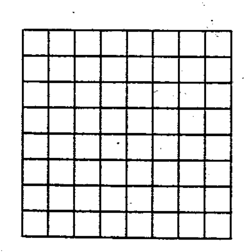
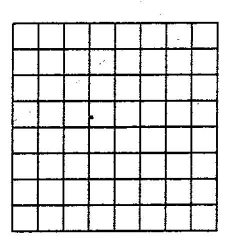
Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
2620	LD H,N	FDCB0586	RES 0,(IY+d)	CBB5	RES 6,L
2A8405	LD HL,(NN)	CB87	RES 0,A	CBBE	RES 7,(HL)
218405	LD HL,NN	CB80	RES 0,B	DDCB05BE	RES 7,(IX+d)
ED47	LD I,A	CB81	RES 0,C	FDCB05BE	RES 7,(IY+d)
DD2A8405	LD IX,(NN)	CB82	RES 0,D	CBBF	RES 7,A
DD218405	LD IX,NN	CB83	RES 0,E	CBB8	RES 7,B
FD2A8405	LD IY,(NN)	CB84	RES 0,H	CBB9	RES 7,C
FD218405	LD IY,NN	CB85	RES 0,L	CBBA	RES 7,D
6E	LD L,(HL)	CB8E	RES 1,(HL)	CBBB	RES 7,E
DD6E05	LD L,(IX+d)	DDCB058E	RES 1,(IX+d)	CBBC	RES 7,H
FD6E05	LD L,(IY+d)	FDCB058E	RES 1,(IY+d)	CBBD	RES 7,L
6F	LD L,A	CB8F	RES 1,A	C9	RET
68	LD L,B	CB88	RES 1,B	D8	RET C
69	LD L,C	CB89	RES 1,C	F8	RET M
6A	LD L,D	CB8A	RES 1,D	D0	RET NC
6B	LD L,E	CB8B	RES 1,E	C0	RET NZ
6C	LD L,H	CB8C	RES 1,H	F0	RET P
6D	LD L,L	CB8D	RES 1,L	E8	RET PE
2E20	LD L,N	CB96	RES 2,(HL)	E0	RET PO
ED7B8405	LD SP,(NN)	DDCB0596	RES 2,(IX+d)	C8	RET Z
F9	LD SP,HL	FDCB0596	RES 2,(IY+d)	ED4D	RETI
DDF9	LD SP,IX	CB97	RES 2,A	ED45	RETN
FDF9	LD SP,IY	CB90	RES 2,B	CB16	RL (HL)
318405	LD SP,NN	CB91	RES 2,C	DDCB0516	RL (IX+d)
EDA8	LDD	CB92	RES 2,D	FDCB0516	RL (IY+d)
EDB8	LDDR	CB93	RES 2,E	CB17	RL A
EDA0	LDI	CB94	RES 2,H	CB10	RL B
EDB0	LDIR	CB95	RES 2,L	CB11	RL C
ED44	NEG	CB9E	RES 3,(HL)	CB12	RL D
00	NOP	DDCB059E	RES 3,(IX+d)	CB13	RL E
B6	OR (HL)	FDCB059E	RES 3,(IY+d)	CB14	RL H
DDB605	OR (IX+d)	CB9F	RES 3,A	CB15	RL L
FDB605	OR (IY+d)	CB98	RES 3,B	17	RLA
B7	OR A	CB99	RES 3,C	CB06	RLC (HL)
B0	OR B	CB9A	RES 3,D	DDCB0506	RLC (IX+d)
B1	OR C	CB9B	RES 3,E	FDCB0506	RLC (IY+d)
B2	OR D	CB9C	RES 3,H	CB07	RLC A
B3	OR E	CB9D	RES 3,L	CB00	RLC B
B4	OR H	CBA6	RES 4,(HL)	CB01	RLC C
B5	OR L	DDCB05A6	RES 4,(IX+d)	CB02	RLC D
F620	OR N	FDCB05A6	RES 4,(IY+d)	CB03	RLC E
EDBB	OTDR	CBA7	RES 4,A	CB04	RLC H
EDB3	OTIR	CBA0	RES 4,B	CB05	RLC L
ED79	OUT (C),A	CBA1	RES 4,C	07	RLCA
ED41	OUT (C),B	CBA2	RES 4,D	ED6F	RLD
ED49	OUT (C),C	CBA3	RES 4,E	CB1E	RR (HL)
ED51	OUT (C),D	CBA4	RES 4,H	DDCB051E	RR (IX+d)
ED59	OUT (C),E	CBA5	RES 4,L	FDCB051E	RR (IY+d)
ED61	OUT (C),H	CBAE	RES 5,(HL)	CB1F	RR A
ED69	OUT (C),L	DDCB05AE	RES 5,(IX+d)	CB18	RR B
D320	OUT (N),A	FDCB05AE	RES 5,(IY+d)	CB19	RR C
EDAB	OUTD	CBAF	RES 5,A	CB1A	RR D
EDA3	OUTI	CBA8	RES 5,B	CB1B	RR E
F1	POP AF	CBA9	RES 5,C	CB1C	RR H
C1	POP BC	CBAA	RES 5,D	CB1D	RR L
D1	POP DE	CBAB	RES 5,E	1F	RRA
E1	POP HL	CBAC	RES 5,H	CB0E	RRC (HL)
DDE1	POP IX	CBAD	RES 5,L	DDCB050E	RRC (IX+d)
FDE1	POP IY	CBB6	RES 6,(HL)	FDCB050E	RRC (IY+d)
F5	PUSH AF	DDCB05B6	RES 6,(IX+d)	CB0F	RRC A
C5	PUSH BC	FDCB05B6	RES 6,(IY+d)	CB08	RRC B
D5	PUSH DE	CBB7	RES 6,A	CB09	RRC C
E5	PUSH HL	CBB0	RES 6,B	CB0A	RRC D
DDE5	PUSH IX	CBB1	RES 6,C	CB0B	RRC E
FDE5	PUSH IY	CBB2	RES 6,D	CB0C	RRC H
CB86	RES 0,(HL)	CBB3	RES 6,E	CB0D	RRC L
DDCB0586	RES 0,(IX+d)	CBB4	RES 6,H	0F	RRCA

Código Objeto	Código Fuente	Código Objeto	Código Fuente	Código Objeto	Código Fuente
ED67	RRD	FDCB05E6	SET 4,(IY+d)	CB3D	SRL L
C7	RST 0	CBE7	SET 4,A	96	SUB (HL)
D7	RST 10H	CBE0	SET 4,B	DD9605	SUB (IX+d)
DF	RST 18H	CBE1	SET 4,C	FD9605	SUB (IY+d)
E7	RST 20H	CBE2	SET 4,D	97	SUB A
EF	RST 28H	CBE3	SET 4,E	90	SUB B
F7	RST 30H	CBE4	SET 4,H	91	SUB C
FF	RST 38H	CBE5	SET 4,L	92	SUB D
CF	RST 8	CBEE	SET 5,(HL)	93	SUB E
9E	SBC A,(HL)	DDCB05EE	SET 5,(IX+d)	94	SUB H
DD9E05	SBC A,(IX+d)	FDCB05EE	SET 5,(IY+d)	95	SUB L
FD9E05	SBC A,(IY+d)	CBEF	SET 5,A	D620	SUB N
9F	SBC A,A	CBE8	SET 5,B	AE	XOR (HL)
98	SBC A,B	CBE9	SET 5,C	DDAE05	XOR (IX+d)
99	SBC A,C	CBEA	SET 5,D	FDAE05	XOR (IY+d)
9A	SBC A,D	CBEB	SET 5,E	AF	XOR A
9B	SBC A,E	CBEC	SET 5,H	A8	XOR B
9C	SBC A,H	CBED	SET 5,L	A9	XOR C
9D	SBC A,L	CBF6	SET 6,(HL)	AA	XOR D
DE20	SBC A,N	DDCB05F6	SET 6,(IX+d)	AB	XOR E
ED42	SBC HL,BC	FDCB05F6	SET 6,(IY+d)	AC	XOR H
ED52	SBC HL,DE	CBF7	SET 6,A	AD	XOR L
ED62	SBC HL,HL	CBF0	SET 6,B	EE20	XOR N
ED72	SBC HL,SP	CBF1	SET 6,C		
37	SCF	CBF2	SET 6,D		
CBC6	SET 0,(HL)	CBF3	SET 6,E		
DDCB05C6	SET 0,(IX+d)	CBF4	SET 6,H		
FDCB05C6	SET 0,(IY+d)	CBF5	SET 6,L		
CBC7	SET 0,A	CBFE	SET 7,(HL)		
CBC0	SET 0,B	DDCB05FE	SET 7,(IX+d)		
CBC1	SET 0,C	FDCB05FE	SET 7,(IY+d)		
CBC2	SET 0,D	CBFF	SET 7,A		
CBC3	SET 0,E	CBF8	SET 7,B		
CBC4	SET 0,H	CBF9	SET 7,C		
CBC5	SET 0,L	CBFA	SET 7,D		
CBCE	SET 1,(HL)	CBFB	SET 7,E		
DDCB05CE	SET 1,(IX+d)	CBFC	SET 7,H		
FDCB05CE	SET 1,(IY+d)	CBFD	SET 7,L		
CBCF	SET 1,A	CB26	SLA (HL)		
CBC8	SET 1,B	DDCB0526	SLA (IX+d)		
CBC9	SET 1,C	FDCB0526	SLA (IY+d)		
CBCA	SET 1,D	CB27	SLA A		
CBCB	SET 1,E	CB20	SLA B		
CBCC	SET 1,H	CB21	SLA C		
CBCD	SET 1,L	CB22	SLA D		
CBD6	SET 2,(HL)	CB23	SLA E		
DDCB05D6	SET 2,(IX+d)	CB24	SLA H		
FDCB05D6	SET 2,(IY+d)	CB25	SLA L		
CBD7	SET 2,A	CB2E	SRA (HL)		
CBD0	SET 2,B	DDCB052E	SRA (IX+d)		
CBD1	SET 2,C	FDCB052E	SRA (IY+d)		
CBD2	SET 2,D	CB2F	SRA A		
CBD3	SET 2,E	CB28	SRA B		
CBD4	SET 2,H	CB29	SRA C		
CBD5	SET 2,L	CB2A	SRA D		
CBDE	SET 3,(HL)	CB2B	SRA E		
DDCB05DE	SET 3,(IX+d)	CB2C	SRA H		
FDCB05DE	SET 3,(IY+d)	CB2D	SRA L		
CBDF	SET 3,A	CB3E	SRL (HL)		
CBD8	SET 3,B	DDCB053E	SRL (IX+d)		
CBD9	SET 3,C	FDCB053E	SRL (IY+d)		
CBDA	SET 3,D	CB3F	SRL A		
CBDB	SET 3,E	CB38	SRL B		
CBDC	SET 3,H	CB39	SRL C		
CBDD	SET 3,L	CB3A	SRL D		
CBE6	SET 4,(HL)	CB3B	SRL E		
DDCB05E6	SET 4,(IX+d)	CB3C	SRL H		

TABLA DE LOS CÓDIGOS DE COLOR PARA TINTA Y FONDO

Color CT + CF -------	Cod. Tinta (CT)		Cod. Fondo (CF)	
	Dec	Hex	Dec	Hex
Color del borde	000	00	00	00
Negro	016	10	01	01
Verde	032	20	02	02
Verde claro	048	30	03	03
Azul oscuro	064	40	04	04
Azul claro	080	50	05	05
Rojo oscuro	096	60	06	06
Azul celeste	112	70	07	07
Rojo	128	80	08	08
Rojo claro	144	90	09	09
Amarillo oscuro	160	A0	10	0A
Amarillo claro	176	B0	11	0B
Verde oscuro	192	C0	12	0C
Magenta	208	D0	13	0D
Gris	224	E0	14	0E
Blanco	240	F0	15	0F

Sprites de 8×8

Sprites de 16×16

